PAŃSTWOWA WYŻSZA SZKOŁA ZAWODOWA W NYSIE
SKRYPT NR 12

Włodzimierz Stanisławski

WPROWADZENIE DO SIECIOWYCH SYSTEMÓW OPERACYJNYCH

OFICYNA WYDAWNICZA PWSZ W NYSIE
NYSA 2006

RECENZENT
Jan Sadecki

SKŁAD I ŁAMANIE
Włodzimierz Stanisławski

KOREKTA
Konrad Szcześniak

PROJEKT GRAFICZNY OKŁADKI
Ryszard Szymończyk

SEKRETARZ OFICYNY
Tomasz Drewniak

© Copyright by
Oficyna Wydawnicza PWSZ w Nysie
Nysa 2006

ISBN 83-60081-10-7

OFICYNA WYDAWNICZA PWSZ W NYSIE
48-300 Nysa, ul. Grodzka 19
tel.: (077) 4090855
e-mail: oficyna@pwsz.nysa.pl
http://www.pwsz.nysa.pl/oficyna

Wydanie I

Druk i oprawa:
Sowa – druk na życzenie
www.sowadruk.pl tel. 022 431-81-40

SPIS TREŚCI

WPROWADZENIE

W książce zawarto zbiór podstawowych wiadomości dotyczących budowy oraz funkcjonowania systemów operacyjnych współczesnych komputerów. Systemy operacyjne są analizowane bardzo ogólnie, bez szczegółów implementacyjnych dotyczących konkretnych rozwiązań. Opisano fundamentalne koncepcje oraz zasady funkcjonowania. Wiele rozwiązań jest ilustrowanych przykładami konkretnych systemów operacyjnych, a szczególnie: UNIX, Windows NT, MS-DOS, NetWare, QNX.

Książka jest w pierwszej kolejności podręcznikiem dla studentów kierunku informatyka, jako pomoc dydaktyczna do wykładu z przedmiotu systemy operacyjne i zawiera podstawowe wiadomości dotyczące funkcjonowania systemów komputerowych. Ponadto książka może być przydatna także dla studentów innych kierunków studiów, a także dla osób zainteresowanych funkcjonowaniem systemów komputerowych. Może być także pomocna dla administratorów systemowych oraz programistów, chcących bardziej efektywnie wykorzystać środki systemowe.

Przedstawiony w książce materiał jest oparty na wieloletniej działalności dydaktycznej autora w Politechnice Opolskiej oraz Państwowej Wyższej Szkole Zawodowej w Nysie.

Pierwsze trzy rozdziały książki wprowadzają czytelnika w problematykę systemów operacyjnych. Po omówieniu podstawowych etapów rozwoju systemów operacyjnych przedstawiono ich klasyfikację oraz ogólną budowę oprogramowania systemowego, ze szczególnym uwzględnieniem budowy wielowarstwowej.

Kolejne trzy rozdziały poświęcono podstawowym zadaniom systemów operacyjnych, związanym z zarządzaniem zasobami lokalnymi komputera: zarządzaniu procesami, pamięcią operacyjną oraz pamięciami masowymi. Szczegółowo zostały omówione wszystkie podstawowe mechanizmy stosowane we współczesnych systemach operacyjnych. Materiał zawarty w tych rozdziałach należy uznać za zasadniczy i zajmuje on decydującą część wykładu.

W ostatnim rozdziale przedstawiono sieciowe funkcje współczesnych systemów operacyjnych. Omówiono koncepcję systemów rozproszonych oraz dwa podstawowe mechanizmy: przekazywania komunikatów oraz wywołania odległej procedury w systemie rozproszonym. Przedstawiono funkcjonowanie sieciowego systemu plików oraz zasady funkcjonowania usługi katalogowej we współczesnych sieciowych systemach operacyjnych.

Chciałbym w tym miejscu wyrazić podziękowanie pani Annie Długosz, aktualnie studentce trzeciego roku kierunku informatyka Politechniki Opolskiej, za wkład pracy związany z przygotowaniem bogatego materiału graficznego książki.

Rozdział I
EWOLUCJA SYSTEMÓW OPERACYJNYCH

Podczas 50-letniego rozwoju systemy operacyjne przeszły bardzo złożoną drogę, wypełnioną licznymi ważnymi odkryciami. Duży wpływ na rozwój systemów operacyjnych miał rozwój elektroniki i architektur systemów komputerowych. Z tego powodu poszczególne etapy rozwoju systemów operacyjnych (SO) są ściśle związane z pojawianiem się nowych platform sprzętowych, takich jak minikomputery, komputery osobiste itp. Szczególny wpływ na ewolucję systemów operacyjnych ma rozwój lokalnych oraz globalnych sieci komputerowych, a zwłaszcza rozwój internetu.

Pierwsze systemy operacyjne pojawiły się w połowie lat 50., gdy rozpoczęto produkcję systemów komputerowych opartych na półprzewodnikach. Dzięki nowym technologiom wzrosła znacznie moc obliczeniowa procesorów a także pojemność pamięci operacyjnych oraz pamięci masowych w stosunku do wcześniejszych komputerów lampowych. Znacznie wzrosła także niezawodność systemów komputerowych, co pozwoliło na ich zastosowanie do rozwiązywania szeregu zadań praktycznych.

Jednocześnie z rozwojem sprzętu komputerowego odbywał się analogiczny proces w zakresie automatyzacji procesu tworzenia programowania oraz organizacji prac obliczeniowych. W tym czasie pojawiły się pierwsze algorytmiczne języki programowania, a co za tym idzie translatory tych języków. Wykonanie każdego programu było związane ze znaczną liczbą czynności, a mianowicie: załadowanie odpowiedniego translatora (np. ALGOL, FORTRAN, COBOL), uruchomienie translatora w celu uzyskania programu binarnego zawierającego kody rozkazów procesora, konsolidacja programu poprzez dołączenie odpowiednich procedur bibliotecznych, załadowanie programu do pamięci operacyjnej, uruchomienie programu, wyprowadzenie wyników na urządzenie zewnętrzne. W ośrodkach obliczeniowych byli zatrudnieni operatorzy wykonujący te czynności dla wszystkich użytkowników ośrodka obliczeniowego.

Niestety stopień wykorzystania procesora w takim systemie był niewielki, gdyż większość czasu procesor oczekiwał na wykonanie odpowiednich prac przez operatora. Przy znacznej wartości komputera uzyskiwano bardzo niską efektywność systemu komputerowego. W celu rozwiązania tego problemu zastosowano przetwarzanie wsadowe, które automatyzowało całą sekwencję operacji związaną z organizacją procesu obliczeniowego. Systemy przetwarzania wsadowego były pierwszymi programami systemowymi, przeznaczonymi do organizacji procesu obliczeniowego, a nie do przetwarzania danych. W celu organizacji procesu obliczeniowego opracowano formalny język sterowania zadaniami, pozwalający programiście określić poszczególne działania oraz ich sekwencję. Operator zestawiał pakiet zadań, które następnie były wykonywane przez system komputerowy bez udziału operatora. Program, który realizował przetwarzanie wsadowe, określany był mianem *monitora* i był

protoplastą współczesnych systemów operacyjnych. Monitor reagował także na najczęściej spotykane sytuacje awaryjne. Wsad zestawiany był najczęściej w postaci pliku kart perforowanych, a także w postaci pliku na taśmie magnetycznej lub dysku magnetycznym.

1.1. Wielozadaniowe systemy operacyjne

Kolejny ważny okres w rozwoju systemów komputerowych to lata 1965-1975. W tym czasie miał miejsce znaczny rozwój w technologii elektronicznej – przejście od pojedynczych tranzystorów do układów scalonych, początkowo małej, następnie średniej oraz wielkiej skali integracji. Dzięki nowym technologiom stała się możliwa budowa złożonych systemów komputerowych (np. IBM/360). W tym czasie opracowano wszystkie podstawowe mechanizmy stosowane we współczesnych systemach operacyjnych, takie jak: wieloprogramowość, wielodostęp, pamięci wirtualne, systemy plików, ochrona danych poprzez określenie praw dostępu, praca w sieci. Jednocześnie nastąpił intensywny rozwój oprogramowania systemowego.

Rewolucyjnym zdarzeniem tego etapu rozwoju systemów komputerowych było praktyczne zastosowanie wieloprogramowości (koncepcja wieloprogramowości była znana już około 10 lat wcześniej). Wieloprogramowość jest sposobem organizacji procesu obliczeniowego, polegającym na jednoczesnym przechowywaniu w pamięci operacyjnej szeregu programów oraz naprzemiennym wykonywaniu tych programów przez jeden procesor komputera. Dzięki takiemu rozwiązaniu znacznie zwiększono stopień wykorzystania procesora. Wieloprogramowość była realizowana w dwu wariantach: w systemach przetwarzania wsadowego oraz w systemach z podziałem czasu.

W systemach wieloprogramowych przetwarzania wsadowego przełączenie wykonywanego programu ma miejsce w momencie, gdy w aktualnie wykonywanym programie rozpoczyna się wykonywanie operacji WE/WY. W wyniku tego rośnie stopień wykorzystania procesora oraz liczba zadań wykonanych w jednostce czasu. Systemy wieloprogramowe z podziałem czasu stosowane są przede wszystkim w systemach wielodostępnych, obsługiwanych za pośrednictwem wielu terminali. W takim przypadku wszyscy użytkownicy mogą jednocześnie wykonywać swoje operacje za pośrednictwem terminala.

Do pierwszych systemów wieloprogramowych z podziałem czasu, opracowanych w połowie lat 60. należy zaliczyć: TSS/360 (IBM), CTSS oraz MULTICS (MIT wraz z Bell Labs oraz General Electric). Systemy wieloprogramowe z podziałem czasu dają użytkownikowi wrażenie wyłącznego korzystania z komputera dzięki temu, że każdy z wykonywanych programów cyklicznie otrzymuje pewną część czasu pracy procesora. Systemy wieloprogramowe z podziałem czasu są mniej efektywne od systemów przetwarzania wsadowego, co stanowi cenę za wygodę pracy użytkowników.

Wielodostęp stosowany był nie tylko w systemach z podziałem czasu, lecz także w systemach wsadowych, w których nie tylko operator lecz także użytkownik miał możliwość tworzenia swojego zadania oraz kierowania jego wykonywaniem. Systemy takie określano mianem systemów zdalnego wprowadzania zadań. Zdarzało się, że poszczególne terminale mogły być umieszczone w znacznej odległości od systemu komputerowego, co wymuszało stosowanie różnego rodzaju połączeń (np. telefonicznych) oraz protokołów komunikacyjnych.

Realizacja wieloprogramowości wymagała rozwiązania szeregu ważnych problemów, do których należy przede wszystkim szybkie przełączenie procesora z jednego programu na drugi, a także ochrona kodu oraz danych poszczególnych programów umieszczonych jednocześnie w pamięci operacyjnej. Jako rozwiązanie tych problemów, w procesorach pojawiły się różne tryby pracy (uprzywilejowany oraz użytkownika), specjalne rejestry służące do szybkiego przełączenia procesora z jednego programu na drugi, mechanizmy ochrony obszarów pamięci a także rozwinięty system przerwań.

W trybie uprzywilejowanym, przewidzianym dla programów wchodzących w skład systemu operacyjnego, procesor mógł wykonywać wszystkie rozkazy, w tym rozkazy ochrony zasobów komputera. Natomiast dla programów użytkowych, pracujących w trybie użytkownika, określona grupa rozkazów była niedostępna. W ten sposób jedynie system operacyjny mógł zarządzać zasobami komputera oraz spełniać rolę monitora i arbitra dla programów użytkowych, które były wykonywane w nieuprzywilejowanym trybie użytkownika. Synchronizacja pracy różnych urządzeń wchodzących w skład komputera, pracujących równolegle i asynchronicznie, takich jak kanały WE/WY, pamięci dyskowe, drukarki itp. odbywa się dzięki systemowi przerwań.

Systemy operacyjne tamtego czasu były bardzo drogie. Np. opracowanie systemu operacyjnego OS/360 o rozmiarze 8 MB kosztowało firmę IBM 80 mln dolarów. Mimo to w przeciągu tego dziesięciolecia został wykonany olbrzymi krok naprzód w rozwoju oprogramowania systemowego, oraz stworzono podwaliny dla współczesnych systemów operacyjnych.

1.2. Systemy operacyjne i globalne sieci komputerowe

Na początku lat 70. pojawiły się pierwsze sieciowe systemy operacyjne, które w odróżnieniu od wielodostępnych systemów operacyjnych, nie tylko pozwoliły na rozproszenie użytkowników, lecz także umożliwiły rozproszoną ochronę i obróbkę danych w wielu komputerach, powiązanych połączeniami elektrycznymi. Sieciowy system operacyjny z jednej strony wykonuje wszystkie funkcje lokalnego systemu operacyjnego, z drugiej charakteryzuje się szeregiem dodatkowych środków, pozwalających na współdziałanie z systemami operacyjnymi pozostałych komputerów w sieci. Moduły programowe

realizujące operacje sieciowe pojawiały się w systemach operacyjnych stopniowo w miarę rozwoju technologii sieciowych.

Mimo że prace teoretyczne związane z koncepcją wzajemnego współdziałania systemów komputerowych prowadzone były niemal od samego pojawienia się komputerów, znaczące wyniki praktyczne dotyczące funkcjonowania sieci komputerowych uzyskano w końcu lat 60. Były to sieci z komutacją pakietów, łączące komputery typu mainfraim oraz superkomputery. W 1969 roku Ministerstwo Obrony Stanów Zjednoczonych zainicjowało prace związane z połączeniem superkomputerów umieszczonych w centrach naukowo-badawczych oraz obronnych we wspólną sieć. Sieć ta, o nazwie ARPANET, stała się pierwowzorem dzisiejszego internetu. Sieć ARPANET łączyła komputery różnych typów, pracujące pod kontrolą różnych systemów operacyjnych.

W 1974 roku firma IBM poinformowała o stworzeniu własnej architektury sieciowej, opracowanej dla swoich komputerów typu mainfraim o nazwie SNA (ang. System Network Architecture). Ta wielopoziomowa architektura jest pod wieloma względami zbliżona do standardowego modelu OSI (ang. Open System Interconnection), który został opracowany nieco później. Sieć SNA pozwalała w globalnej sieci na współdziałanie typu „terminal-terminal", „terminal-komputer" oraz „komputer-komputer". Najniższe poziomy architektury SNA były realizowane sprzętowo, natomiast wyższe – programowo.

W tym samym czasie w Europie prowadzono intensywne prace związane z opracowaniem standardu sieci X.25. Standard ten został przyjęty w 1974 roku, a firma IBM wprowadziła obsługę protokołów X.25 do architektury SNA oraz do swoich systemów operacyjnych.

1.3. Systemy operacyjne minikomputerów oraz pierwsze sieci lokalne

W połowie lat 70. zaczęto szeroko stosować systemy minikomputerowe, takie jak PDP-11, Nova, HP. Architektura minikomputerów była znacznie uproszczona w porównaniu z komputerami typu mainfraim, co znalazło odbicie w ich systemach operacyjnych. Ze względu na ograniczone zasoby minikomputerów niewielkie były także możliwości systemów operacyjnych stosowanych w tego rodzaju komputerach.

Ważnym etapem w historii minikomputerów, a także w historii systemów operacyjnych, było opracowanie wielozadaniowego, z podziałem czasu, systemu operacyjnego UNIX. Początkowo system ten był stosowany w minikomputerze PDP-7, a następnie PDP-11, a w drugiej połowie lat 70. powszechnie jako wielozadaniowy system operacyjny.

W tym czasie 90% kodu systemu UNIX było napisane w języku C, a szerokie zastosowanie efektywnych kompilatorów języka C pozwoliło na łatwe przenoszenie systemu UNIX na różne typy komputerów. Mimo że UNIX opracowany był dla minikomputerów, to takie cechy jak elastyczność,

elegancja, duże możliwości funkcjonalne, a także otwartość (znany kod źródłowy), zadecydowały o zastosowaniu systemu UNIX w superkomputerach, komputerach typu mainfraim, serwerach oraz stacjach roboczych i komputerach osobistych.

Dostępność minikomputerów i stosunkowo niska cena oraz ich szerokie zastosowanie w przedsiębiorstwach, spowodowały zapotrzebowanie na sieci lokalne. Przedsiębiorstwo mogło sobie pozwolić na zakup kilku minikomputerów umieszczonych w jednym budynku, co wywołało zapotrzebowanie na wymianę danych między nimi.

Pierwsze lokalne sieci komputerowe tworzono w oparciu o szeregowe porty WE/WY. W systemie UNIX pierwszym programem służącym do obsługi sieci lokalnej był UUCP (ang. UNIX-to-UNIX Copy Program), dostępny w wersji 7. AT&T UNIX z 1978 roku. Program ten pozwalał kopiować pliki z jednego komputera na drugi z zastosowaniem interfejsu RS-232.

1.4. Rozwój systemów operacyjnych w latach 80.

Do najważniejszych zdobyczy tego dziesięciolecia należy zaliczyć: opracowanie stosu TCP/IP jako podstawy Internetu, standaryzację technologii lokalnych sieci komputerowych a także opracowanie systemów operacyjnych komputerów osobistych.

Pierwszy roboczy wariant stosu protokołów TCP/IP powstał w końcu lat 70. Stos ten zawierał zbiór uniwersalnych protokołów dla różnorodnego środowiska obliczeniowego i był przeznaczony do połączenia eksperymentalnej sieci ARPANET z innymi sieciami. W 1983 roku stos protokołów TCP/IP został przyjęty przez Ministerstwo Obrony Stanów Zjednoczonych jako standard. Przejście sieci ARPANET do stosu TCP/IP znacznie przyspieszyła realizacja stosu TCP/IP w systemie BSD UNIX (interfejs gniazd – ang. sockets). Od tego momentu wszystkie wersje systemu UNIX zawierały realizację stosu TCP/IP, a UNIX stał się sieciowym systemem operacyjnym.

Wdrożenie protokołów TCP/IP w sieci ARPANET dało tej sieci wszystkie podstawowe cechy współczesnego Internetu. W 1983 roku sieć ARPANET rozdzielono na dwie części: MILNET - przeznaczoną dla celów obronnych, oraz nową, ARPANET, określoną następnie mianem Internet. Dzięki niezależności protokołów TCP/IP od platformy sprzętowej i systemowej, elastyczności i efektywności, a także otwartości i dostępności do standardów, stały się one nie tylko głównym mechanizmem transportowym Internetu, lecz także głównym elementem składowym sieciowych systemów operacyjnych.

W latach 80. pojawiło się wiele komercyjnych wersji systemu UNIX: SunOS, HP-UX, Irix, AIX oraz wiele innych. Różnorodność wersji systemu UNIX spowodowała problem kompatybilności. W celu jego rozwiązania opracowano standardy, takie jak POSIX oraz XPG, określające interfejs systemu operacyjnego dla programów użytkowych, a firma AT&T opracowała

kilka wersji UNIX (System III oraz System V) w celu konsolidacji środowisk rozwijających UNIX na poziomie jądra systemu.

Początek lat 80. jest związany z jeszcze jednym bardzo istotnym w historii systemów operacyjnych zdarzeniem – pojawieniem się komputerów osobistych. Z punktu widzenia architektury komputery osobiste nie różniły się od minikomputerów, lecz były znacznie tańsze. Jeśli minikomputer był używany przez oddział firmy czy uniwersytet, to komputery osobiste używane były przez poszczególnych użytkowników. Ponieważ komputery te były używane nie tylko przez specjalistów, lecz także przez użytkowników nieznających zasad programowania, systemy operacyjne tych komputerów musiały zawierać bardziej przyjazny interfejs użytkownika. Ponadto komputery osobiste zaczęto stosować na szeroką skalę, co wymusiło znaczny rozwój sieci komputerowych, a oprogramowanie sieciowe stało się podstawowym składnikiem systemów operacyjnych.

Jednak wprowadzenie przyjaznego interfejsu użytkownika oraz funkcji sieciowych do systemów operacyjnych komputerów osobistych odbywało się stopniowo. Pierwsza wersja popularnego systemu operacyjnego DOS nie posiadała tych możliwości. Był to jednozadaniowy system operacyjny z interfejsem znakowym. Głównymi zadaniami systemu było zarządzanie plikami umieszczonymi na dyskach twardych lub elastycznych oraz sekwencyjne wykonywanie programów. Ponieważ procesor 8086/8088 (Intel) nie mógł pracować w trybie uprzywilejowanym, więc MS-DOS nie był chroniony przed programami użytkownika. Brakujące funkcje systemu operacyjnego były uzupełniane programami zewnętrznymi, jak np. interfejs w postaci programu Norton Commander. Największy wpływ na rozwój oprogramowania dla komputerów osobistych miał system operacyjny Windows, który w początkowej fazie był nakładką na system operacyjny DOS.

Funkcje sieciowe były także realizowane w postaci nakładek na systemy operacyjne. Podczas pracy sieciowej niezbędna jest praca wielozadaniowa, przy której użytkownik interaktywny wykonuje swoje operacje, a pozostali – poprzez sieć – uzyskują dostęp do zasobów komputera. W takim przypadku od systemu operacyjnego oczekuje się choćby podstawowej obsługi pracy wielozadaniowej. Dla systemu DOS pierwsze funkcje związane z obsługą sieci pojawiły się w wersji 3.1, a dotyczyły blokowania plików. Dzięki nim system operacyjny mógł zapewnić jednoczesny dostęp do plików wielu użytkownikom.

Inną drogę wybrała firma Novell – opracowała nowy system operacyjny o nazwie NetWare, z wbudowanymi funkcjami sieciowymi, który na wiele lat stał się wzorcem wydajności, niezawodności oraz bezpieczeństwa w sieci lokalnej. Pierwsza wersja systemu NetWare opracowana dla komputerów osobistych pojawiła się w 1983 roku i była przeznaczona dla komputerów z procesorami rodziny 8086. System operacyjny NetWare stosowany był przede wszystkim jako centralny serwer w sieci i pełnił rolę serwera plików, zapewniając bardzo szybki zdalny dostęp do plików oraz ich bezpieczeństwo.

W 1987 roku, w wyniku wspólnych prac firm Microsoft i IBM, opracowano pierwszy wielozadaniowy system operacyjny dla komputerów z procesorem 80286, wykorzystujący chroniony tryb pracy – system OS/2. W systemie zastosowano wielozadaniowość z wywłaszczeniem, wielowątkowość, pamięci wirtualne, graficzny interfejs użytkownika oraz maszynę wirtualną do wykonywania programów napisanych dla systemu DOS. System operacyjny OS/2 wraz z funkcjami wielozadaniowości oraz systemem plików HPFS (ang. Hyper Performance File System) i systemem ochrony plików okazał się bardzo dobrą platformą dla lokalnych sieci komputerowych z komputerami osobistymi. Szczególne znaczenie miały powłoki sieciowe LAN Manager (Microsoft) oraz LAN Server (IBM), pozwalające stworzyć z komputera jednocześnie serwer plików oraz stację roboczą.

Wspólne prace firm Microsoft oraz IBM doprowadziły do powstania bardzo popularnego protokołu transportowego oraz jednocześnie interfejsu programisty dla sieci lokalnych – NetBIOS. Interfejs ten był stosowany praktycznie we wszystkich systemach operacyjnych komputerów osobistych, a także stosowany jest we współczesnych systemach operacyjnych.

W latach 80. przyjęto podstawowe standardy dla technologii komunikacyjnych w sieciach lokalnych: w 1980 roku – Ethernet, w 1985 – Token Ring, w końcu lat 80. – FDDI. Pozwoliło to na zapewnienie kompatybilności sieciowych systemów operacyjnych na najniższych poziomach oraz na standaryzację interfejsów sieciowych. Poza systemami operacyjnymi opracowanymi wyłącznie dla komputerów osobistych (DOS, NetWare, OS/2) zaczęto adaptować istniejące systemy operacyjne do komputerów osobistych. Przykładem może być system operacyjny UNIX, opracowany przez firmę Santa Cruz Operation (SCO UNIX).

1.5. Osobliwości współczesnego etapu rozwoju systemów operacyjnych

W latach 90. praktycznie wszystkie komercyjne systemy operacyjne stały się systemami sieciowymi. Funkcje sieciowe zostały wbudowane w jądro systemu i stały się jego nieodłączną częścią. Współczesne systemy operacyjne zostały wyposażone we wszystkie podstawowe technologie tak lokalnych (Ethernet, Fast Ethernet, Gigabit Ethernet, Token Ring, FDDI, ATM), jak i rozległych sieci (X.25, frame relay, ISDN, ATM), a także IP, IPX, AppleTaalk, RIP, OSPF, NLSP. W systemach operacyjnych wykorzystuje się możliwości multipleksowania szeregu protokołów sieciowych, dzięki czemu komputery mogą jednocześnie współpracować z różnymi klientami i serwerami. Pojawiły się także wyspecjalizowane systemy operacyjne przeznaczone wyłącznie do wykonywania zadań komunikacyjnych. Przykładem takiego systemu jest IOS firmy Cisco Systems, organizujący wielozadaniowe wykonywanie programów realizujących protokoły komunikacyjne.

W drugiej połowie lat 90. wszyscy producenci systemów operacyjnych znacznie zwiększyli wspomaganie współpracy z Internetem (wyjątkiem UNIX,

w którym funkcje te były wykonywane od dawna). Poza samym stosem TCP/IP systemy operacyjne wspomagają takie protokoły warstwy zastosowań, jak telnet, ftp, DNS itp. Dzięki temu komputer zmienił swoje funkcje: z czysto obliczeniowych na komunikacyjne z rozwiniętymi możliwościami obliczeniowymi.

Podczas całego minionego dziesięciolecia szczególną uwagę zwracano na korporacyjne sieciowe systemy operacyjne. Rozwój systemów korporacyjnych jest także najistotniejszym zadaniem współczesnym. Systemy te wyróżniają się możliwością stabilnej pracy w wielkich sieciach, charakterystycznych dla dużych firm, posiadających swe oddziały w wielu miastach a także państwach. Sieci takie charakteryzują się znacznym stopniem heterogeniczności sprzętu oraz oprogramowania, co powoduje, że korporacyjny sieciowy system operacyjny musi efektywnie współpracować z różnymi systemami operacyjnymi i różnymi platformami sprzętowymi. Współcześnie można wyróżnić trzy systemy operacyjne mające znaczenie w systemach korporacyjnych: Novell NetWare 4.x i 5.x, Microsoft Windows NT, a także UNIX.

Dla korporacyjnych systemów operacyjnych szczególnie ważnym zagadnieniem jest możliwość centralnego administrowania i zarządzania systemem, z wykorzystaniem wspólnej bazy danych, w której zapisano wszystkie informacje o użytkownikach, komputerach, urządzeniach komunikacyjnych, a także o oprogramowaniu zainstalowanym na poszczególnych komputerach. W systemie Windows taką rolę pełni Active Directory, natomiast w NetWare – NDS. Opracowanie wielofunkcyjnej, skalowalnej usługi zarządzania siecią korporacyjną jest dzisiaj strategicznym kierunkiem rozwoju sieciowych systemów operacyjnych. Od tego zależy także w dużej części rozwój Internetu.

Na współczesnym etapie rozwoju systemów operacyjnych na pierwszy plan wychodzą środki zapewnienia bezpieczeństwa. Jest to związane z rosnącą wartością informacji przetwarzanej przez komputery, a także rosnącym poziomem zagrożeń występujących podczas przesyłania danych sieciami publicznymi. Systemy operacyjne charakteryzują się rozwiniętymi środkami ochrony informacji, opartymi na szyfrowaniu danych, uwierzytelnianiu oraz autoryzacji.

W ostatnim czasie miał miejsce znaczny rozwój interfejsu człowiek – komputer. Efektywność pracy człowieka jest podstawowym czynnikiem określającym efektywność całego systemu obliczeniowego. Wysiłek człowieka nie powinien być tracony na dobór parametrów procesu obliczeniowego, jak to miało miejsce w poprzednich systemach operacyjnych. Współczesny system operacyjny bierze na siebie wykonanie zadań wyboru parametrów środowiska operacyjnego, wykorzystując do tego różne algorytmy adaptacyjne. Dla przykładu tzw. time-out w protokołach komunikacyjnych często określa się w zależności od warunków pracy sieci. Rozdział pamięci operacyjnej między procesy wykonywany jest automatycznie z wykorzystaniem mechanizmów

pamięci wirtualnej, w zależności od aktywności tych procesów i informacji o częstości wykorzystania poszczególnych stron pamięci. Chwilowe priorytety procesów określane są dynamicznie w zależności od historii, biorąc pod uwagę czas oczekiwania procesu w kolejce, procent wykorzystania przydzielonego kwantu czasu, intensywność operacji WE/WY.

W sposób ciągły jest udoskonalana interaktywna praca z komputerem poprzez włączanie do systemu operacyjnego rozbudowanych interfejsów graficznych, wykorzystujących także dźwięk oraz wideo. Jest to szczególnie ważne dla przekształcenia komputera w terminal nowej sieci publicznej.

Rozdział II
ZADANIA SYSTEMÓW OPERACYJNYCH

Obecnie istnieje cały szereg systemów operacyjnych różniących się funkcjonalnością, obszarami zastosowań, platformami sprzętowymi oraz metodami realizacji. Spośród znacznej liczby realizowanych przez nie zadań opisano te, które są charakterystyczne dla wszystkich systemów operacyjnych.

2.1. Systemy operacyjne dla komputerów autonomicznych

System operacyjny komputera stanowi zbiór wzajemnie powiązanych programów, funkcjonujących jako interfejs między użytkownikiem i aplikacjami z jednej strony i sprzętem komputera z drugiej. Z tego punktu widzenia system operacyjny wykonuje dwie grupy funkcji:

- udostępnianie użytkownikowi oraz programiście maszyny wirtualnej w miejsce udostępniania konkretnych zasobów sprzętowych komputera,
- zwiększanie efektywności wykorzystania komputera poprzez racjonalne zarządzanie jego zasobami.

2.1.1. System operacyjny jako maszyna wirtualna

W celu efektywnego wykorzystania komputera użytkownik nie musi znać wszystkich szczegółów funkcjonowania poszczególnych podzespołów komputera, a także rozkazów procesora. Użytkownik/programista chciałby wykorzystywać wysokopoziomowe funkcje systemu operacyjnego. Np. podczas pracy z dyskiem programiście oraz użytkownikowi wystarczy reprezentacja pamięci masowej w postaci odpowiednio zorganizowanego systemu plików. Natomiast informacje o numerze powierzchni dyskowej, numerach ścieżek oraz sektorów, na których zapisano dane wchodzące w skład pliku, jak również organizacja fizycznego zapisu danych na powierzchni magnetycznej dysku, sposób synchronizacji obracającego się dysku ze sterownikiem dyskowym czy aktualne położenie głowicy dyskowej, są informacjami nieistotnymi dla użytkownika/programisty. Wynika z tego, że system operacyjny ukrywa przed użytkownikiem cały szereg szczegółów dotyczących funkcjonowania zasobów sprzętowych komputera, dając możliwość prostej i wygodnej pracy.

System operacyjny zwalnia użytkownika/programistę nie tylko z konieczności bezpośredniej pracy z zasobami sprzętowymi komputera, lecz także bierze na siebie wszystkie rutynowe operacje związane ze sterowaniem urządzeniami zewnętrznymi. W rezultacie, rzeczywisty komputer wykonujący stosunkowo niewielki zbiór operacji elementarnych określonych przez listę rozkazów procesora, zostaje przekształcony w maszynę wirtualną wykonującą

szeroki zbiór złożonych funkcji. Analogicznie jak komputer rzeczywisty, maszyna wirtualna jest także sterowana instrukcjami, lecz instrukcje te należą do znacznie wyższego poziomu, jak np. usunięcie pliku o podanej nazwie, uruchomienie określonego programu, zmiana priorytetu zadania, wydrukowanie zawartości pliku itp. W ten sposób zadaniem systemu operacyjnego jest udostępnianie użytkownikowi/programiście maszyny wirtualnej, na której nieporównanie łatwiej wykonywać operacje niż w przypadku bezpośredniego korzystania z zasobów komputera.

2.1.2. System operacyjny jako zarządca zasobów

System operacyjny nie tylko udostępnia użytkownikom i programistom wygodny interfejs do sprzętowych zasobów komputera, lecz także pełni rolę zarządcy udostępniającego zasoby komputera. Do podstawowych zasobów współczesnych systemów komputerowych mogą być zaliczone: procesory, pamięć operacyjna, liczniki czasowe, zbiory danych, dyski, pamięci taśmowe, drukarki, urządzenia sieciowe oraz szereg innych. Zasoby są udostępniane przez system operacyjny poszczególnym procesom. Pojęcie *procesu* (zadania) jest podstawowym pojęciem we współczesnych systemach operacyjnych, a w najprostszy sposób można je zdefiniować jako program w trakcie jego realizacji, wraz z przydzielonymi mu zasobami. Pod pojęciem *program* rozumiemy obiekt statyczny, złożony z kodu oraz danych, przechowywany w pliku dyskowym. Natomiast *proces* to obiekt dynamiczny, tworzony przez system operacyjny jako następstwo uruchomienia programu przez użytkownika lub system operacyjny. Utworzony proces staje się podstawową jednostką wykonawczą systemu komputerowego.

Zarządzanie zasobami systemu komputerowego w celu najbardziej efektywnego ich wykorzystania jest głównym zadaniem systemu operacyjnego. Dla przykładu wielozadaniowy system operacyjny organizuje jednoczesne wykonywanie szeregu procesów poprzez kolejne przydzielanie im procesora, eliminując w ten sposób przestoje procesora wywołane operacjami WE/WY. System operacyjny rozwiązuje także konflikty wynikające z jednoczesnego odwoływania się wielu procesów do tego samego zasobu. Kryteria efektywności, na podstawie których system operacyjny organizuje zarządzanie zasobami komputera, mogą być bardzo różne (np. przepustowość systemu komputerowego lub czas reakcji komputera).

Podczas zarządzania zasobami, zależnie od rodzaju zasobu, system operacyjny wykonuje następujące operacje:

- planowanie zasobu, czyli określenie któremu procesowi, kiedy oraz w jakiej części (w przypadku gdy zasób może być udostępniany częściami) będzie przydzielony dany zasób,
- zaspokajanie zapotrzebowania na zasoby,
- śledzenie stanu oraz wykorzystania zasobu, czyli przechowywanie informacji o tym, czy dany zasób jest w danym momencie dostępny,

- rozwiązywanie konfliktów między procesami.

W celu rozwiązania tych ogólnych zadań zarządzania zasobami systemy operacyjne stosują różne algorytmy, które określają takie cechy całego systemu operacyjnego, jak charakterystyki wydajności, obszar zastosowania a także interfejs użytkownika. Dla przykładu zastosowany algorytm zarządzania procesorem w znacznym stopniu określa czy system operacyjny może być zastosowany jako system z podziałem czasu, system przetwarzania wsadowego czy system czasu rzeczywistego.

Zadanie organizacji efektywnego współbieżnego wykorzystania zasobów przez szereg procesów jest zadaniem bardzo złożonym. Wynika to przede wszystkim z przypadkowego odwoływania się procesów do zasobów komputera. W systemie wielozadaniowym z każdym zasobem związana jest kolejka, w której oczekują procesy odwołujące się do danego zasobu. System operacyjny obsługuje te kolejki na podstawie różnych algorytmów: w kolejności zgłoszenia, na podstawie priorytetów, obsługi cyklicznej itd.

Zarządzanie zasobami systemu komputerowego jest bardzo złożonym zadaniem systemu operacyjnego, a w szczególności systemu wielozadaniowego. Większość funkcji zarządzania zasobami jest wykonywana wyłącznie przez oprogramowanie systemowe, bez udziału programów użytkowych.

2.1.3. Funkcjonalne części składowe systemów operacyjnych komputerów autonomicznych

Funkcje systemu operacyjnego komputera autonomicznego z zasady grupuje się bądź na podstawie lokalnych zasobów komputera, bądź na podstawie zadań wykonywanych na wszystkich zasobach. Takie grupy funkcji określa się mianem podsystemów. Do najważniejszych podsystemów zarządzania poszczególnymi zasobami komputera należy zaliczyć: podsystemy zarządzania procesami, pamięcią, plikami i urządzeniami WE/WY. Natomiast do podsystemów ogólnych – podsystem interfejsu użytkownika, ochrony danych czy administrowania.

Zarządzanie procesami

Najistotniejszym podsystemem systemu operacyjnego, bezpośrednio wpływającym na funkcjonowanie systemu komputerowego, jest podsystem zarządzania procesami. Dla każdego nowo utworzonego procesu system operacyjny tworzy odpowiednie struktury danych, w których przechowywane są informacje o zapotrzebowaniu procesu na zasoby komputera oraz o rzeczywiście przydzielonych mu zasobach. Aby proces mógł być wykonany, system operacyjny musi przydzielić mu pamięć operacyjną, w której będą przechowywane kod oraz dane, a także udostępnić procesowi niezbędną część czasu procesora. Ponadto procesowi należy umożliwić dostęp do takich zasobów jak pliki oraz urządzenia WE/WY.

Do informacyjnych struktur procesu są włączane dodatkowe dane określające historię procesu w systemie operacyjnym (np. jaką część czasu proces zużył na wykonywanie operacji WE/WY a jaką na obliczenia), aktualny stan procesu (aktywny czy zablokowany), aktualny priorytet procesu itp. W wielozadaniowym systemie operacyjnym ma miejsce jednoczesna realizacja wielu procesów. Część procesów została utworzona w odpowiedzi na żądania użytkowników – procesy takie określane są mianem *procesów użytkowych*. Natomiast część procesów, określana mianem *procesów systemowych*, została powołana do życia przez system operacyjny w celu realizacji określonych zadań systemowych.

Ważnym zadaniem systemu operacyjnego jest ochrona zasobów przydzielonych danemu procesowi przed pozostałymi procesami. Do najpilniej strzeżonych zasobów procesu należy przydzielona procesowi pamięć operacyjna, w której przechowywany jest kod realizowanego programu oraz przetwarzane dane. Zbiór wszystkich obszarów pamięci operacyjnej przydzielonych procesowi określa się mianem *przestrzeni adresowej procesu*. Można powiedzieć, że każdy proces posiada własną, wydzieloną przestrzeń adresową.

Chronione są także inne zasoby, takie jak pliki, urządzenia WE/WY itd. System operacyjny może nie tylko chronić zasoby przydzielone wyłącznie jednemu procesowi, lecz także organizować wspólne wykorzystanie zasobów przez wiele procesów, jak np. segment pamięci współdzielonej.

W całym cyklu życia procesu jego wykonywanie może być wielokrotnie przerywane oraz wznawiane. Aby była możliwa kontynuacja realizacji procesu po przerwaniu jego wykonywania jest niezbędne odtworzenie jego środowiska operacyjnego. Środowisko operacyjne procesu jest określone przez: stan rejestrów procesora wraz z licznikiem rozkazów, wskaźniki na otwarte w procesie pliki, informacja o niezakończonych operacjach WE/WY, kody błędów przy wywołaniach usług systemowych itd. Stan środowiska operacyjnego procesu określany jest mianem *kontekstu procesu*, a podczas przełączania procesu wykonywana jest operacja przełączenia kontekstu. System operacyjny wykonuje także operacje związane z *synchronizacją procesów*, pozwalające na wstrzymanie biegu procesu do momentu wystąpienia określonego zdarzenia w systemie.

W systemie operacyjnym nie ma bezpośredniego związku między procesami a programami. Jeden program przechowywany w określonym pliku może być wykonywany w wielu równolegle realizowanych procesach, a dany proces może wykonać operację załadowania nowego programu przechowywanego w dowolnym pliku. W przypadku złożonych aplikacji może okazać się korzystne, aby programy były realizowane w wielu równoległych procesach, które komunikują się między sobą i wymieniają dane. W tym celu system operacyjny udostępnia cały szereg mechanizmów *komunikacji międzyprocesowej*.

Głównymi zadaniami systemu operacyjnego w zakresie zarządzania procesami są: planowanie przydziału procesora poszczególnym procesom, synchronizacja procesów oraz obsługa operacji komunikacji między równolegle wykonywanymi procesami.

Zarządzanie pamięcią operacyjną

Pamięć operacyjna jest dla procesu równie ważnym zasobem komputera jak procesor, ponieważ aby proces mógł być realizowany, musi być umieszczony w pamięci operacyjnej. Zarządzanie pamięcią jest związane przede wszystkim z: udostępnieniem istniejącej pamięci fizycznej wszystkim procesom występującym w danej chwili w systemie, załadowaniem kodu oraz danych do przydzielonych miejsc pamięci, modyfikacją rozkazów w załadowanych programach, w których występują adresy fizyczne a także ochroną obszarów pamięci poszczególnych procesów.

Istnieje szereg algorytmów udostępniania pamięci poszczególnym procesom. Różnią się one między innymi: liczbą ciągłych bloków pamięci przydzielonych jednemu procesowi (od jednego do wielu), możliwością przemieszczania granicy obszaru pamięci (granica ta może być niezmienna na cały czas trwania procesu, lub może być swobodnie przemieszczania podczas przydzielania procesowi dodatkowego obszaru pamięci). W wielu systemach operacyjnych proces otrzymuje określoną liczbę stron pamięci o ściśle określonym i niezmiennym rozmiarze, lecz możliwy jest także przydział segmentu o zmiennym rozmiarze.

We współczesnych systemach operacyjnych zarządzanie pamięcią operacyjną odbywa się poprzez zastosowanie mechanizmów tzw. *pamięci wirtualnej*. Zastosowanie pamięci wirtualnych pozwala użytkownikowi oraz programiście wykonywać operacje w taki sposób, jakby programy oraz dane przechowywane były w pojedynczych ciągłych blokach pamięci o rozmiarze często przewyższającym rozmiar pamięci fizycznej komputera. Podczas przenoszenia programów oraz danych między pamięcią operacyjną i pamięcią masową, podsystem pamięci wirtualnej wykonuje operacje translacji adresów wirtualnych, wyznaczonych na etapie kompilacji i konsolidacji programu, na adresy fizyczne określające numery komórek w pamięci operacyjnej. Wszystkie operacje przenoszenia danych oraz translacji adresów są wykonywane przez system operacyjny i są „przeźroczyste” dla użytkownika/programisty.

Ochrona pamięci związana jest z kontrolą adresów i ma za zadanie niedopuszczenie do zapisu a także odczytu komórek pamięci wchodzących do przestrzeni adresowej danego procesu przez inne procesy. Takie operacje występują najczęściej na skutek błędów w programach.

W celu właściwego zarządzania pamięcią operacyjną system operacyjny wykonuje następujące operacje: przechowuje informacje o aktualnie zajętych oraz wolnych obszarach pamięci, przydziela pamięć nowym procesom oraz zwalnia pamięć po procesach, które zakończyły działanie, wykonuje operacje związane z ochroną pamięci, przenosi procesy między pamięcią operacyjną

a pamięcią masową w przypadku, gdy rozmiar pamięci operacyjnej jest zbyt mały, wykonuje operacje związane z translacją adresów wirtualnych na adresy fizyczne.

Zarządzanie plikami oraz urządzeniami WE/WY

Jednym z podstawowych podsystemów systemu operacyjnego jest *system plików*. Dzięki funkcjonowaniu tego podsystemu dane, które przechowywane są w pamięci dyskowej na określonych powierzchniach, ścieżkach oraz w sektorach, dla użytkownika są dostępne w postaci pliku – struktury danych zawierającej ciąg bajtów o zmiennej długości, identyfikowanej nazwą. Dla wygody użytkownika pliki grupowane są w katalogi. System plików pozwala użytkownikowi na wykonywanie określonych operacji na plikach i katalogach, takich jak: poszukiwanie po nazwie, usunięcie, wyprowadzenie zawartości na urządzenie zewnętrzne, zmiana zawartości itp.

W celu reprezentacji znacznej liczby zbiorów danych, rozrzuconych po różnych cylindrach i powierzchniach pamięci dyskowej, współczesne systemy operacyjne stosują hierarchiczną strukturę plików i katalogów. System operacyjny na podstawie nazwy pliku wyznacza fizyczny adres danych na dysku (powierzchnia dysku, cylinder, sektor), a także organizuje jednoczesny dostęp do plików i zabezpiecza przed nieautoryzowanym dostępem do plików. Podczas operacji na plikach system plików ściśle współpracuje z podsystemem sterującym urządzeniami zewnętrznymi (podsystem WE/WY), który na żądanie systemu plików wykonuje przesłania danych między dyskami a pamięcią operacyjną.

Podsystem WE/WY spełnia rolę interfejsu do wszystkich urządzeń zewnętrznych przyłączonych do komputera. Zbiór tych urządzeń jest bardzo liczny: dyski twarde, elastyczne oraz optyczne, drukarki, skanery, monitory, plotery, modemy, karty sieciowe oraz szereg bardziej specjalizowanych urządzeń WE/WY, jak np. przetworniki A/C i C/A. Poszczególne urządzenia mogą różnić się zbiorem rozkazów, przy pomocy których wykonuje się operacje przesłań danych, szybkością pracy, kodowaniem przesyłanych danych, możliwością pracy równoległej i wieloma innymi szczegółami.

Program sterujący konkretnym modelem urządzenia zewnętrznego i uwzględniający wszystkie szczegóły dotyczące tego urządzenia określany jest mianem sterownika (ang. driver). Każdy model urządzenia WE/WY musi być wyposażony we własny sterownik. Natomiast system operacyjny musi zapewnić precyzyjnie zdefiniowany interfejs między sterownikami urządzeń a pozostałymi składnikami systemu operacyjnego. Dzięki temu producenci urządzeń zewnętrznych mogą opracować sterowniki do swoich urządzeń dla poszczególnych systemów operacyjnych. Twórcy programów użytkowych mogą także bezpośrednio wykorzystywać interfejsy urządzeń zewnętrznych, jednak ze względu na niezbędną znajomość znacznej liczby szczegółów, takie niskopoziomowe operacje są rzadko stosowane.

Jednym z głównych zadań systemu operacyjnego jest udostępnienie wysokopoziomowego, uniwersalnego interfejsu dla obsługi urządzeń WE/WY. Poczynając od systemu operacyjnego UNIX, taki uniwersalny interfejs WE/WY jest zbudowany na podstawie koncepcji dostępu do pliku. Koncepcja ta polega na tym, że wymiana danych z dowolnym urządzeniem WE/WY jest wykonywana analogicznie jak wymiana danych z plikiem posiadającym nazwę i będącym zbiorem bajtów.

Ochrona danych oraz administracja

Pierwszą linią ochrony danych systemu operacyjnego przed nieautoryzowanym dostępem jest procedura wejścia logicznego. System operacyjny powinien upewnić się, że użytkownik pragnący rozpocząć pracę posiada zgodę administratora. Funkcje ochrony systemu operacyjnego są ściśle związane z funkcjami administrowania, ponieważ administrator określa uprawnienia użytkowników kontrolowane podczas ich odwoływania się do zasobów systemowych (plików, katalogów, drukarek itp.). Poza tym administrator systemowy ogranicza możliwości użytkowników do wykonywania określonych operacji w systemie. Np. można zabronić użytkownikowi wykonywania operacji zamknięcia oraz restartu systemu, przestawiania zegara, kończenia zadań innych użytkowników, zmiany uprawnień dostępu do określonych katalogów oraz plików.

Ważnym środkiem ochrony danych są funkcje monitorowania systemu operacyjnego, polegające na rejestracji wszystkich zdarzeń, od których zależy bezpieczeństwo systemu. Dla przykładu udane i nieudane próby logicznego wejścia do systemu, operacje dostępu do wybranych katalogów oraz plików, wykorzystywania drukarek itp.

Interfejs programowy (ang. Application Programming Interface - API)

Programiści aplikacji użytkowych mogą wykorzystywać w swoich programach odwołania do systemu operacyjnego w celu wykonania określonych operacji. Dla przykładu we współczesnych systemach operacyjnych wszystkie operacje związane z urządzeniami WE/WY mogą być wykonywane wyłącznie przez oprogramowanie systemowe. Poza tym programista może wykorzystywać szereg funkcji usługowych systemu operacyjnego znacznie upraszczających tworzenie oprogramowania użytkowego. Funkcje usługowe realizują uniwersalne operacje często wykonywane w programach użytkowych (np. operacje na łańcuchach tekstowych). Zbiór funkcji systemu operacyjnego, udostępnianych programiście tworzącemu oprogramowanie użytkowe, nosi nazwę interfejsu programowego API. Dla końcowego użytkownika funkcje interfejsu API są ukryte za powłoką znakowego lub graficznego interfejsu użytkownika.

Dla twórców aplikacji użytkowych wszystkie szczegóły funkcjonowania konkretnego systemu operacyjnego są zawarte w interfejsie API. Oznacza to, że systemy operacyjne o różnej organizacji wewnętrznej, lecz jednakowym zbiorze

funkcji interfejsu API, dla programów aplikacyjnych prezentują się jako identyczne. Umożliwia to standaryzację systemów operacyjnych oraz zapewnia możliwość przenoszenia aplikacji między różnymi systemami operacyjnymi. Np. spełnianie ogólnego standardu API UNIX (jednym z takich standardów jest POSIX) przez wszystkie wersje systemu UNIX pozwala mówić o uogólnionym systemie UNIX, mimo że poszczególne wersje różnią się znacznie pod względem budowy wewnętrznej.

Aplikacje wykonują odwołania do funkcji API za pomocą wywołań systemowych mających taką samą postać jak wywołania podprogramów, a przekazywanie danych odbywa się najczęściej przez rejestry procesora oraz przez stos.

Interfejs użytkownika

Poza interfejsem programów użytkowych system operacyjny tworzy wygodny interfejs dla użytkownika (końcowy użytkownik, programista, administrator) wykonującego operacje w systemie. Współczesne systemy operacyjne udostępniają zaawansowane funkcje interfejsu użytkownika do interaktywnej pracy przy dwu rodzajach terminali: alfanumerycznym (znakowym) oraz graficznym.

Do pracy przy terminalu znakowym system operacyjny udostępnia użytkownikowi zbiór poleceń odzwierciedlających możliwości systemu operacyjnego. Z zasady system operacyjny udostępnia polecenia pozwalające na wykonywanie następujących operacji: uruchamianie i zatrzymywanie programów użytkowych, operacje na plikach i katalogach, pobieranie informacji o aktualnym stanie systemu (np. liczba wykonywanych procesów, wolny obszar na dysku itp.), a także administrowanie systemem. Polecenia mogą być wykonywane nie tylko w trybie interaktywnym, lecz także w trybie wsadowym (plik tekstowy, zawierający polecenia do wykonania). Tak w trybie interaktywnym jak i wsadowym polecenia wykonywane są przez program określany mianem interpretera poleceń.

Wprowadzanie poleceń może być znacznie uproszczone jeśli system operacyjny jest wyposażony w graficzny interfejs użytkownika (ang. GUI – Graphic User Interface). W takim przypadku użytkownik, najczęściej przy pomocy myszki, wybiera z menu właściwe opcje.

2.2. Sieciowe systemy operacyjne

Sieć komputerowa to zbiór komputerów powiązanych połączeniami sieciowymi, wyposażonych w odpowiednie oprogramowanie pozwalające użytkownikom na dostęp do zasobów wszystkich komputerów wchodzących w skład sieci. W sieci mogą współpracować komputery różnych typów (komputery osobiste, stacje robocze, superkomputery itp.). System komunikacyjny może zawierać różne rodzaje mediów transmisyjnych, rozgałęźniki sygnałów, przełączniki, routery i inne urządzenia zapewniające

transmisję danych między dowolnymi komputerami w sieci. Sieć komputerowa pozwala wykorzystywać zasoby lokalnego komputera oraz dodatkowo daje możliwość dostępu do informacyjnych oraz sprzętowych zasobów innych komputerów w sieci.

Sieciowy system operacyjny pełni rolę interfejsu między użytkownikiem a niskopoziomowymi szczegółami sprzętowo-programowych zasobów sieci. Np. zamiast używać liczbowych adresów komputerów w sieci, takich jak adresy MAC lub IP, sieciowy system operacyjny umożliwia stosowanie symbolicznych nazw komputerów. Dzięki temu sieć komputerowa, wraz z szeregiem złożonych i zagmatwanych szczegółów, dla użytkownika jest przedstawiona w postaci uporządkowanego zbioru zasobów informacyjno-sprzętowych.

2.2.1. Sieciowe a rozproszone systemy operacyjne

W zależności od maszyny wirtualnej, jaką tworzy system operacyjny, zbiór komputerów połączonych sieciami może pełnić rolę sieciowego lub rozproszonego systemu operacyjnego. Sieciowy SO tworzy dla użytkownika wirtualny system obliczeniowy, w którym nie jest przed nim ukrywana informacja o rozproszeniu systemu, a oprogramowanie systemowe pozwala na wykorzystywanie zasobów całego systemu. Można go określić mianem *wirtualnej sieci.* W takim systemie użytkownik wykorzystuje zasoby sieciowe, a dostęp do nich wymaga określonych operacji. Użytkownicy sieci wirtualnej muszą wiedzieć, gdzie znajdują się określone dane (np. pliki), a także w jawny sposób wykonywać określone operacje przesłań plików.

Domyślnie zadania użytkownika uruchamiane są na komputerze lokalnym. Poza tym użytkownik sieciowego SO może uruchomić zadanie na innym komputerze w sieci, jednak zawsze jest świadomy na którym. W takim przypadku użytkownik może postąpić dwojako:

- wykonać logiczne przyłączenie się do odległego komputera wykorzystując polecenie typu *remote login*,
- wykonać polecenie odległego wykonania programu, wskazując komputer, na którym ma być wykonany program.

Głównym kierunkiem rozwoju sieciowych systemów operacyjnych jest osiągnięcie możliwie wysokiego stopnia przeźroczystości zasobów sieciowych. W idealnym przypadku sieciowy system operacyjny powinien udostępniać zasoby sieciowe w postaci zasobów jednego scentralizowanego komputera wirtualnego. Taki system operacyjny określany jest mianem *rozproszonego SO*. Rozproszony system operacyjny w sposób dynamiczny przydziela zadania dla poszczególnych komputerów wchodzących w skład systemu, tworząc użytkownikowi wirtualny obraz jednego komputera. W takim przypadku przed użytkownikiem ukrywana jest informacja o strukturze systemu, topologii sieci itp.

2.2.2. Dwa znaczenia terminu „Sieciowy SO”

Sieciowy system operacyjny może być traktowany jako zbiór systemów operacyjnych poszczególnych komputerów wchodzących w skład sieci, przy czym na komputerach sieci mogą funkcjonować te same lub odmienne systemy operacyjne. Np. bardzo często spotykane są sieci, w których znajdują się komputery zarządzane przez systemy operacyjne UNIX, NetWare oraz Windows. Poszczególne systemy operacyjne funkcjonują niezależnie od siebie w tym znaczeniu, że każdy z nich podejmuje autonomiczne decyzje o tworzeniu i zatrzymaniu procesów oraz zarządzaniu własnymi zasobami. Systemy operacyjne muszą obsługiwać określony zbiór protokołów sieciowych służących do komunikowania się procesów wykonywanych na poszczególnych węzłach sieci oraz do udostępniania zasobów dla innych użytkowników sieci.

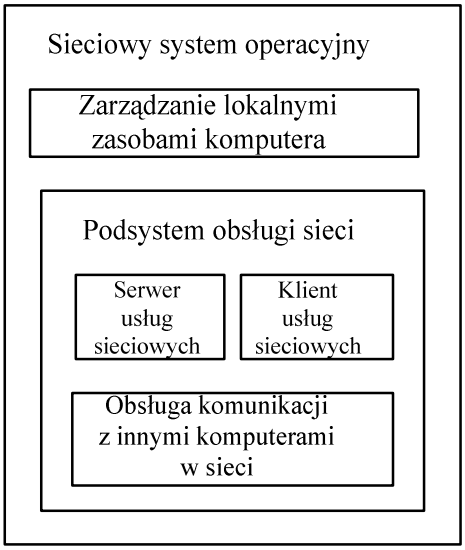
Z drugiej strony, system operacyjny komputera umożliwiającego pracę w sieci, czyli udostępniającego swoje zasoby innym komputerom pracującym w sieci, a także posiadającego zdolność wykorzystywania zasobów innych komputerów, określany jest także sieciowym systemem operacyjnym.

W ten sposób pojęcie „sieciowy system operacyjny” posiada dwa znaczenia: jako zbiór systemów operacyjnych wszystkich komputerów w sieci oraz jako system operacyjny danego komputera, który umożliwia pracę w sieci.

2.2.3. Funkcjonalne części składowe sieciowego SO

Do podstawowych części składowych sieciowego SO należy zaliczyć (rys. 2.1):

- *podsystem zarządzania lokalnymi zasobami komputera* realizujący wszystkie funkcje SO autonomicznego komputera (zarządzanie pamięcią operacyjną, planowanie i szeregowanie procesów, zarządzanie pamięcią masową, interfejs użytkownika itp.),
- *podsystem obsługi sieci* pracujący w architekturze klient/serwer, do którego można zaliczyć następujące komponenty:
  - serwer usług sieciowych udostępniający lokalne zasoby komputera oraz określone usługi innym użytkownikom sieci,
  - klient usług sieciowych pozwalający na wykorzystywanie zasobów innych komputerów w sieci,
  - oprogramowanie pozwalające na komunikację z innymi komputerami w sieci.



Rys. 2.1. Funkcjonalne części składowe sieciowego systemu operacyjnego

Upraszczając, pracę sieciowego SO można przedstawić następująco: załóżmy, że użytkownik komputera A postanowił zapisać swój plik w pamięci masowej komputera B. W tym celu wykonuje odpowiednie polecenie na komputerze A, które następnie jest przekazywane do klienta usług sieciowych tego komputera. W następnej kolejności za pośrednictwem sieci jest wysyłany odpowiedni komunikat do komputera B, w którym serwer usług sieciowych przyjmuje polecenie wraz z zawartością pliku i zleca oprogramowaniu zarządzającemu lokalnymi zasobami komputera wykonanie odpowiednich operacji. Serwer usług sieciowych na komputerze B musi nieustannie pracować i oczekiwać na zgłoszenia od innych komputerów w sieci.

Oprogramowanie obsługujące komunikację w sieci tworzy odpowiednie komunikaty, dzieli komunikaty na części, takie jak pakiety czy ramki, przetwarza nazwy komputerów na odpowiednie adresy numeryczne, obsługuje proces niezawodnego dostarczenia komunikatów, w przypadku sieci o złożonej topologii określa drogę przekazania komunikatu itp. Zasady komunikowania się komputerów w sieci określone są przy pomocy protokołów komunikacyjnych, takich jak np. Ethernet, Token Ring, IP, IPX itd. Aby dany komputer mógł komunikować się z różnymi komputerami w sieci, oprogramowanie pozwalające na komunikację w niej musi obsługiwać cały szereg protokołów komunikacyjnych.

2.2.4. Usługi sieciowe

Współdziałanie serwera i klienta umożliwiające dostęp do konkretnego zasobu komputera przez sieć, określa się mianem *usługi sieciowej*. Usługa sieciowa udostępnia użytkownikom sieci cały zbiór operacji. Każda usługa jest związana z określonym typem zasobu sieciowego oraz z określonym sposobem dostępu do tego zasobu. Usługa drukowania zapewnia dostęp użytkownikom sieci do drukarek sieciowych, a usługa pocztowa umożliwia przesyłanie poczty elektronicznej.

Spośród wielu usług sieciowych można wyróżnić usługi zorientowane nie tyle na użytkownika sieci, co na administratora, jak np. usługa *Bindery* w systemie NetWare, pozwalająca administratorowi wykorzystywać bazę danych o użytkownikach sieci. Bardziej zaawansowaną usługą sieciową jest usługa katalogowa polegająca na utworzeniu bazy danych nie tylko o użytkownikach, lecz także o wszystkich programowych i sprzętowych komponentach sieci. Typowymi przykładami tej usługi są DNS firmy Novell oraz StreetTalk firmy Banyan. Innymi przykładami usług sieciowych przeznaczonych dla administratora są usługi monitorowania sieci, bezpieczeństwa, tworzenia kopii zapasowych i archiwizowania itp. Zestaw usług sieciowych oferowanych przez system operacyjny decyduje o pozycji danego systemu w szerokiej rodzinie sieciowych systemów operacyjnych.

Usługi sieciowe są skonstruowane zgodnie z architekturą klient/serwer. Dla każdej usługi funkcjonuje serwer, pełniący bierną rolę w tej architekturze i oczekujący na zgłoszenia klientów. Klient usługi inicjuje połączenie z odpowiednim serwerem i pełni aktywną rolę w architekturze klient/serwer. W systemie operacyjnym komputera przyłączonego do sieci mogą funkcjonować jednocześnie serwer i klient danej usługi sieciowej, lub tylko jeden z nich. Ogólnie można powiedzieć, że serwer udostępnia klientowi swoje zasoby, natomiast klient może z nich korzystać.

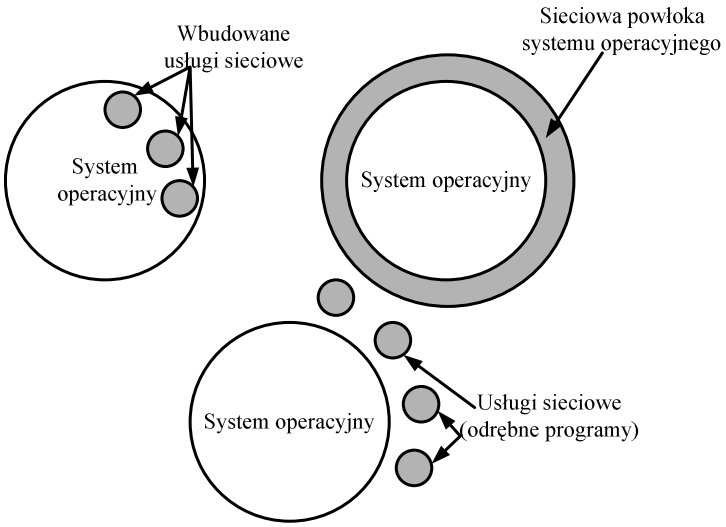
2.2.5. Wbudowane usługi sieciowe oraz powłoki sieciowe

Z praktycznego punktu widzenia istnieje kilka sposobów budowy sieciowych systemów operacyjnych, różniących się „głębokością" umieszczenia usług sieciowych w system operacyjny (rys. 2.2):

- usługi sieciowe są głęboko wbudowane w system operacyjny,
- usługi sieciowe są zgromadzone w powłoce systemu operacyjnego,
- usługi sieciowe są opracowane i zainstalowane jako odrębne oprogramowanie.

W pierwszych sieciowych systemach operacyjnych większość usług sieciowych była zgrupowana w powłoce nadbudowanej na lokalnym systemie operacyjnym. Jedynie niewielka liczba podstawowych funkcji sieciowych była wbudowana w lokalny system operacyjny (przykładem może być system operacyjny LAN Manager stanowiący powłokę na systemie OS/2).

W większości współczesnych sieciowych systemów operacyjnych usługi sieciowe są wbudowane w system operacyjny już na etapie projektu systemu, co zapewnia logiczną spójność oraz wysoką wydajność. Przykładami takich systemów mogą być: Windows, UNIX, NetWare, OS/2 Warp.



Rys. 2.2. Warianty budowy sieciowych systemów operacyjnych

Inną możliwością realizacji usług sieciowych jest umieszczenie ich w powłoce systemu operacyjnego, udostępniającego wszystkie podstawowe operacje związane z zarządzaniem zasobami komputera. W takim przypadku programy realizujące usługi sieciowe działają w środowisku lokalnego systemu operacyjnego. Ze względu na funkcje realizowane przez daną powłokę można wyróżnić powłoki realizujące funkcje klienta usług sieciowych oraz serwera. Np. system operacyjny MS-DOS z nałożoną powłoką kliencką NetWare był stosowany jako podstawowy system operacyjny stacji roboczych w sieci NetWare. Natomiast przykładem powłok realizujących funkcje serwera usług sieciowych mogą być LAN Server i LAN Manager, NetWare dla SO UNIX oraz File and Print Services dla NetWare.

Z danym rodzajem zasobu mogą być związane usługi różniące się między sobą protokołem komunikacyjnym między klientem a serwerem. Np. wbudowana usługa plikowa systemu Windows NT realizuje protokół o nazwie SMB stosowany we wszystkich systemach operacyjnych Microsoft, a dodatkowa usługa plikowa wchodząca w skład powłoki File and Print Services for NetWare dla systemu Windows wykorzystuje protokół NCP stosowany przez Novell. Ponadto w skład systemu Windows NT wchodzi serwer FTP realizujący usługi serwera plików w oparciu o protokół FTP. Ponadto nic nie stoi na przeszkodzie, aby wprowadzić do systemu inne usługi plikowe, jak np. NFS.

Trzeci sposób implementacji usług sieciowych polega na opracowaniu odrębnego produktu. Przykładem może być serwer zdalnego zarządzania WinFrame (firma Citrix) przeznaczony do pracy w systemie Windows NT. Oprogramowanie to uzupełnia możliwości wbudowanego w Windows NT serwera zdalnego dostępu *Remote Access Server*.

2.3. Sieciowe systemy operacyjne typu peer-to-peer oraz systemy z wyróżnionymi serwerami

W zależności od podziału funkcji w sieci, poszczególne komputery mogą pełnić jedną z trzech ról:

- *serwer* obsługujący wyłącznie zgłoszenia od klientów,
- *klient* korzystający z usług udostępnianych przez komputery pełniące rolę serwerów,
- serwer i klient jednocześnie.

Jest oczywiste, że sieć nie może składać się wyłącznie z serwerów lub wyłącznie z klientów. Sieć spełniająca swoją rolę i zapewniająca wzajemne współdziałanie komputerów może być zbudowana na jeden z następujących trzech sposobów:

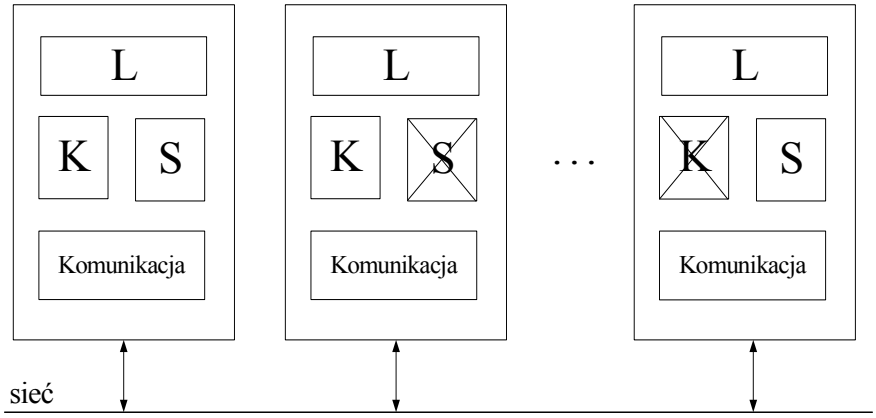
- sieć oparta na równorzędnych komputerach, pełniących jednocześnie rolę serwerów i klientów, udostępniających sobie wzajemnie określone usługi sieciowe – sieć peer-to-peer,
- sieć z wyróżnionymi komputerami, pełniącymi rolę serwerów usług sieciowych,
- sieć posiadająca cechy obydwu wcześniejszych typów – sieć hybrydowa.

2.3.1. Sieciowe systemy operacyjne typu peer-to-peer

W sieci typu peer-to-peer wszystkie komputery posiadają takie same możliwości i realizują jednakowe usługi sieciowe, mając jednakowy dostęp do wszystkich zasobów sieci. Każdy z użytkowników może udostępnić dowolny zasób swojego komputera innym użytkownikom sieci. W sieci wszystkie komputery zarządzane są przez ten sam system operacyjny i posiadają potencjalnie te same możliwości. Jest oczywiste, że na każdym komputerze w sieci funkcjonują tak serwery jak i klienty poszczególnych usług sieciowych (rys. 2.3). Przykładami sieciowych systemów operacyjnych typu peer-to-peer mogą być: LANtastic, Personal Ware, Windows NT itd.

Przy potencjalnym równouprawnieniu wszystkich komputerów, w sieci typu peer-to-peer w rzeczywistości często ma miejsce asymetria, gdyż niektórzy użytkownicy nie chcą udostępniać swoich zasobów pozostałym użytkownikom sieci. W takim przypadku serwery odpowiednich usług sieciowych w ich systemach operacyjnych pozostają nieaktywne, a komputery pełnią rolę „czystych" klientów. Administrator systemu ma także uprawnienia do

wyłączenia możliwości klienckich danego komputera w sieci, dzięki czemu komputer taki staje się „czystym” serwerem.



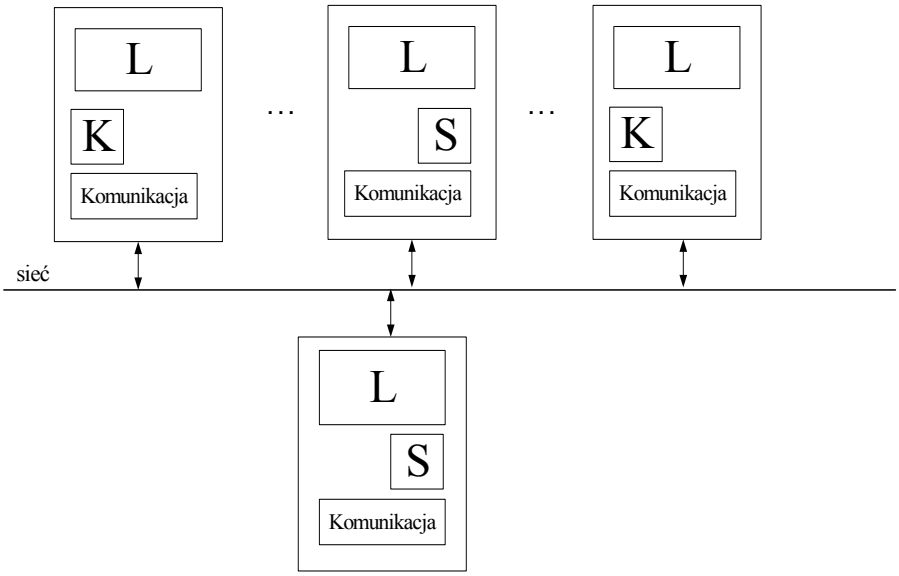
Rys. 2.3. Sieć typu peer-to-peer. L – zarządzanie zasobami lokalnymi, K – klient usługi sieciowej, S – serwer usługi sieciowej

2.3.2. Sieciowe systemy operacyjne z wyróżnionymi serwerami

W rozbudowanych sieciach komputerowych wydzielenie komputerów, które pełnią rolę serwerów usług sieciowych, daje możliwość przetwarzania danych skupionych na serwerach, a także możliwość skutecznej ochrony. Uproszczony schemat takiej sieci przedstawiono na rys. 2.4. System operacyjny, który przystosowany jest wyłącznie do pracy jako serwer usług sieciowych, może pracować znacznie efektywniej, niż miało to miejsce w poprzednim przypadku. Ma to szczególne znaczenie w rozbudowanych sieciach, w których setki (a nawet tysiące) użytkowników korzystają z zasobów serwera. W takim przypadku serwer sieciowy powinien być zainstalowany na komputerze o odpowiedniej mocy obliczeniowej, a system operacyjny powinien spełniać wysokie wymagania co do niezawodności i wydajności.

Im mniej funkcji realizuje system operacyjny serwera, tym efektywniej może je realizować. Powoduje to, że producenci sieciowych systemów operacyjnych świadomie rezygnują z określonych funkcji systemowych. Dobrym przykładem może być serwer w systemie NetWare. Twórcy systemu zoptymalizowali przede wszystkim obsługę systemu plików oraz usługę drukowania. Z tego powodu usunięto z systemu operacyjnego cały szereg elementów istotnych dla uniwersalnego systemu operacyjnego, takich jak: graficzny interfejs użytkownika, ochrona pamięci w warunkach pracy wielozadaniowej, mechanizmy pamięci wirtualnej. Wszystko to pozwoliło na uzyskanie wysokiej szybkości operacji na plikach, co zapewnia, że system NetWare od wielu lat jest bardzo szeroko stosowany.

Jednakże wąska specjalizacja sieciowych systemów operacyjnych przeznaczonych do pracy w charakterze serwera usług sieciowych stanowi także słabą stronę tych systemów. Np. brak w systemie NetWare uniwersalnego interfejsu programisty (API) oraz środków ochrony aplikacji nie pozwala na wykorzystanie systemu operacyjnego jako środowiska do wykonywania aplikacji. Powoduje to, że w przypadku gdy będzie istniała konieczność wykonywania innych operacji sieciowych niż obsługa systemu plików oraz drukowania, w sieci muszą się znaleźć inne komputery pełniące rolę serwerów usług sieciowych.

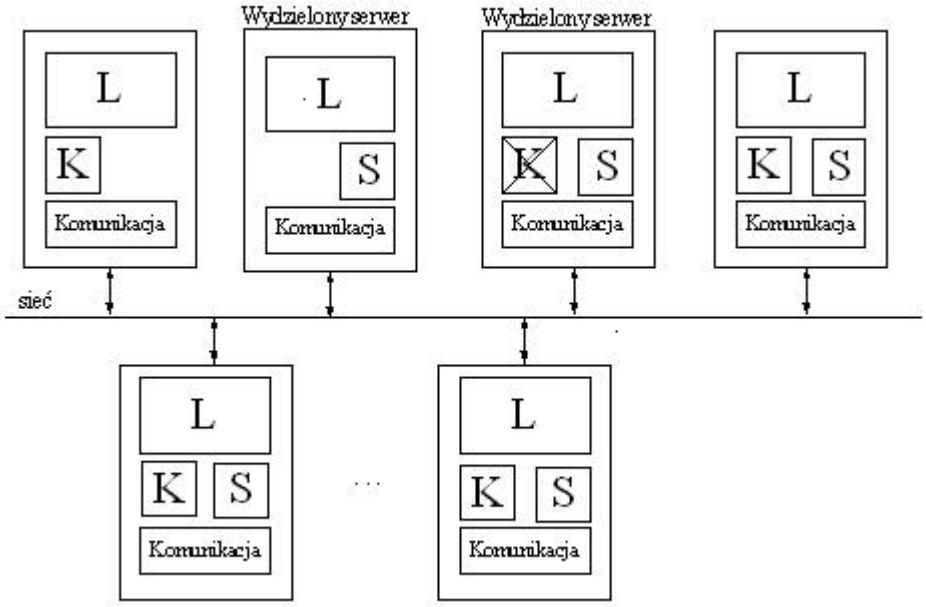


Rys. 2.4. Sieć z wydzielonymi serwerami

Z tego powodu w wielu sieciowych systemach operacyjnych odchodzi się od funkcjonalnego ograniczania serwera, a do oprogramowania serwera wprowadza się wszystkie komponenty pozwalające wykorzystywać komputer tak jako uniwersalny serwer, jak również jako komputer kliencki. Takie serwery są wyposażone w graficzny interfejs użytkownika oraz w interfejs API. Cechy te zbliżają sieciowe systemy operacyjne z wyróżnionymi serwerami do systemów peer-to-peer, jednak można wskazać szereg różnic, które odróżniają te systemy operacyjne od systemów peer-to-peer:

- wspomaganie wieloprocesorowych platform sprzętowych,
- wspomaganie jednoczesnego wykonywania wielu procesów oraz połączeń sieciowych,
- włączenie do systemu operacyjnego komponentów centralnego administrowania siecią (np. usługi informacyjnej czy usługi uwierzytelniania i autoryzacji użytkowników sieci),
- szeroki wachlarz usług sieciowych.

Klienckie wersje sieciowych systemów operacyjnych z wydzielonymi serwerami z zasady nie zawierają funkcji serwerowych, co znacznie upraszcza ich budowę. Natomiast szczególną uwagę kładzie się na interfejs użytkownika oraz na usługi klienckie. Najprostsze sieciowe systemy klienckie dostarczają jedynie usługę obsługi plików oraz usługę drukowania. Producenci sieciowych systemów operacyjnych dostarczają z zasady dwa warianty danego systemu operacyjnego. Pierwszy wariant jest przeznaczony do pracy w charakterze serwera, natomiast drugi – w charakterze klienta. Obydwa warianty są oparte na tym samym kodzie bazowym, jednak różnią się zbiorem udostępnianych usług oraz parametrami konfiguracji. W wielu współczesnych systemach operacyjnych istnieje możliwość łączenia związków typu klient/serwer między komputerami sieci ze związkami typu peer-to-peer, co daje możliwość tworzenia systemów hybrydowych. Przykład takiego systemu przedstawiono na rys. 2.5.



Rys. 2.5. Sieć hybrydowa

2.4. Wymagania stawiane współczesnym sieciowym systemom operacyjnym

Od sieciowego systemu operacyjnego wymaga się przede wszystkim sprawnego zarządzania zasobami sieci, a także wygodnego interfejsu użytkownika oraz interfejsu API. Współczesny system operacyjny musi być systemem wielozadaniowym, a także obsługiwać pamięć wirtualną, okienkowy graficzny interfejs użytkownika oraz udostępniać cały szereg innych usług.

Poza tym przed sieciowymi systemami operacyjnymi stawia się następujące wymagania eksploatacyjne:

- *Rozszerzalność.* W przeciwieństwie do sprzętu komputerowego, który starzeje się bardzo szybko, czas życia systemów operacyjnych może być mierzony dziesiątkami lat (np. UNIX ma ponad 30 lat). W tak długim czasie systemy operacyjne ulegają przemianom, dostosowując się do zmieniających się wymagań, przy czym zmiany te są często głębsze niż zmiany sprzętu komputerowego. Zmiany systemów operacyjnych najczęściej dotyczą obsługi nowych urządzeń zewnętrznych lub nowych technologii sieciowych. Rozszerzalny system operacyjny daje możliwość łatwej rozbudowy bez konieczności dokonywania zmian w systemie. Jest to możliwe dzięki modułowej budowie oprogramowania systemowego, w której system operacyjny zbudowany jest z odrębnych modułów powiązanych ze sobą precyzyjnie zdefiniowanymi interfejsami,
- *Mobilność.* Oprogramowanie systemowe powinno mieć możliwość przenoszenia z jednej platformy sprzętowej na inną (zmiana platformy sprzętowej związana jest nie tylko ze zmianą procesora, lecz także ze zmianą organizacji całego komputera). Mobilne systemy operacyjne posiadają szereg wariantów realizacji dla różnych platform sprzętowych (systemy takie określa się mianem wieloplatformowych),
- *Kompatybilność.* Współcześnie istnieje wiele systemów operacyjnych (wersje systemu UNIX, MS-DOS, Windows 3.x, Windows NT, OS/2), dla których na przestrzeni wielu lat opracowano znaczną liczbę aplikacji. Z tego powodu dla użytkownika zmieniającego system operacyjny jest bardzo istotne, aby dana aplikacja pracowała także w nowym systemie,
- *Niezawodność i odporność na zawieszenia.* System operacyjny powinien być odporny na błędy (tak wewnętrzne jak i zewnętrzne). Działanie systemu powinno także być zawsze przewidywalne, a programy aplikacyjne nie mogą mieć wpływu na destabilizację systemu operacyjnego. Niezawodność oraz odporność na zawieszenia jest zapewniona przede wszystkim przez odpowiednią architekturę, leżącą u podstaw budowy systemu, oraz przez jakość oprogramowania,
- *Bezpieczeństwo.* Współczesny system operacyjny powinien chronić dane oraz inne zasoby systemu komputerowego przed nieautoryzowanym dostępem. W tym celu system operacyjny powinien udostępniać następujące środki: uwierzytelnianie użytkowników (określenie ich legalności), autoryzacja, czyli przydzielenie legalnym użytkownikom określonych uprawnień dostępu do zasobów komputera, audyt, czyli rejestracja wszystkich „podejrzanych" zdarzeń dla bezpieczeństwa systemu. W sieciowych systemach operacyjnych konieczne jest ponadto zapewnienie bezpieczeństwa danych przesyłanych sieciami,

- *Wydajność*. System operacyjny powinien cechować się możliwie dużą szybkością oraz krótkim czasem reakcji na polecenia użytkownika, w takim stopniu, w jakim pozwala na to platforma sprzętowa. Na wydajność systemu ma wpływ wiele czynników. Do najważniejszych należy zaliczyć architekturę systemu operacyjnego, różnorodność funkcji, jakość oprogramowania systemowego, możliwość implementacji systemu operacyjnego na wydajnej platformie sprzętowej.

Rozdział III
ARCHITEKTURA SYSTEMU OPERACYJNEGO

Pojęcie „system” jest powszechnie używane w technice i oznacza dowolny układ techniczny charakteryzujący się znaczną złożonością. Jednak jednoznaczne zdefiniowanie tego pojęcia jest w zasadzie niemożliwe. Główną cechą systemu jest duża liczba wzajemnie powiązanych i współdziałających elementów (podsystemów). Każdy z elementów może być z kolei rozpatrywany jako system zawierający szereg powiązanych elementów. Wynika z tego, że system posiada strukturę hierarchiczną (każdy element systemu sam jest systemem z szeregiem podsystemów) – rys. 3.1. W przypadku systemu operacyjnego, będącego tworem programowym, poszczególne podsystemy (określane mianem modułów) są programami spełniającymi określone funkcje.

Dowolny system złożony powinien charakteryzować się racjonalną strukturą, czyli zawierać szereg podsystemów posiadających w pełni określone przeznaczenie z precyzyjnie ustalonymi regułami współdziałania z innymi podsystemami. Ścisłe określenie zadań każdego modułu znacznie upraszcza modyfikację oraz rozwój systemu. W przeciwnym wypadku system złożony bez dobrze zdefiniowanej struktury łatwiej opracować od nowa niż modernizować.

Złożoność funkcjonalna systemu operacyjnego nieuchronnie prowadzi do złożoności architektonicznej, pod którą rozumie się organizację strukturalną, określającą powiązania poszczególnych modułów programowych. Z zasady w skład systemu operacyjnego wchodzą moduły wykonawcze oraz obiektowe o formatach standardowych dla danego systemu operacyjnego, biblioteki różnych typów, moduły tekstu źródłowego programów, moduły programowe specjalnego formatu (np. program ładujący systemu operacyjnego, sterowniki WE/WY), pliki konfiguracyjne, pliki zawierające dokumentację itp.

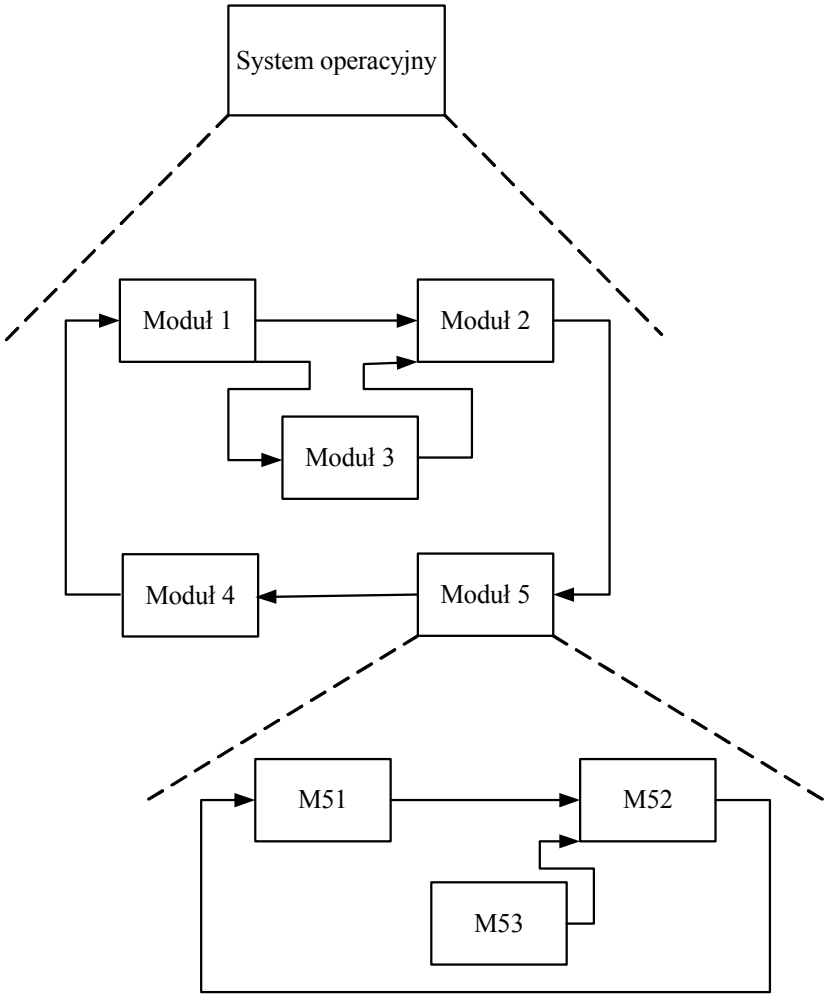
Większość współczesnych systemów operacyjnych posiada dobrze zdefiniowaną strukturę, pozwalającą na rozbudowę oraz przenoszenie na inne platformy sprzętowe.

3.1. Jądro oraz moduły pomocnicze systemu operacyjnego

Najogólniej w systemie operacyjnym można wyróżnić dwie grupy modułów:

- jądro, zawierające moduły wykonujące podstawowe funkcje systemu operacyjnego,
- moduły wykonujące funkcje pomocnicze.

Moduły jądra wykonują podstawowe funkcje systemu operacyjnego, takie jak zarządzanie procesami, pamięcią, urządzeniami WE/WY itp. i stanowią rdzeń systemu operacyjnego, bez którego nie może on realizować swoich podstawowych funkcji.



Rys. 3.1. Hierarchiczna budowa systemu

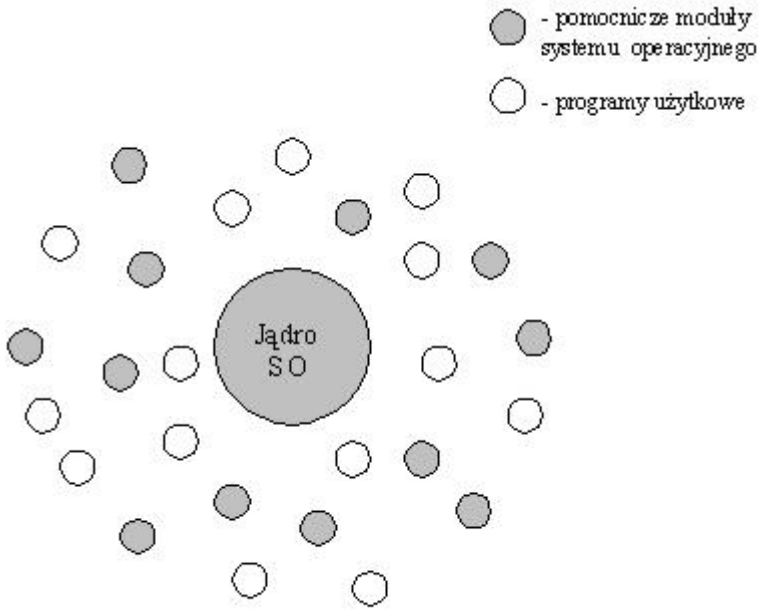
W skład jądra wchodzą dwa rodzaje funkcji:

- funkcje wewnątrzsystemowe,
- funkcje interfejsu aplikacji użytkowych API (ang. Application Program Interface).

Funkcje wewnątrzsystemowe realizują zadania organizacji procesu obliczeniowego, takie jak przełączanie kontekstu czy obsługa przerwań, i są niedostępne dla aplikacji. Druga klasa funkcji jądra służy do wspomagania aplikacji, udostępniając tzw. programowe środowisko użytkownika. W celu wykonywania określonych operacji aplikacje mogą wywoływać funkcje jądra poprzez tzw. wywołania systemowe, jak np. otwarcie i odczyt pliku, wyprowadzenie informacji na monitor ekranowy, odczyt czasu systemowego itp. Funkcje jądra, które mogą być wywoływane przez aplikacje, tworzą tzw.

interfejs użytkownika API. Ponieważ funkcje jądra systemu są najczęściej wykonywanymi programami, więc szybkość wykonywania tych funkcji decyduje o wydajności całego systemu operacyjnego. W celu zapewnienia dużej szybkości wywoływania funkcji systemowych wszystkie moduły jądra, lub znaczna ich część, przechowywane są w pamięci operacyjnej, tworząc tzw. rezydentną część systemu operacyjnego. Można powiedzieć, że jądro systemu operacyjnego jest siłą napędową wszystkich procesów obliczeniowych w systemie komputerowym.

Poza jądrem w skład systemu operacyjnego wchodzą moduły realizujące różnego rodzaju operacje pomocnicze, jak np. operacje archiwizacji danych, defragmentacji dysków, edycji plików itp. Funkcje te realizowane są najczęściej przez programy aplikacyjne, co powoduje, że trudno jest zdefiniować ostrą granicę między systemem operacyjnym a programami użytkowymi (rys. 3.2).



Rys. 3.2. Brak ostrych granic między systemem operacyjnym a programami użytkowymi

W skład funkcji pomocniczych można zaliczyć następujące grupy:

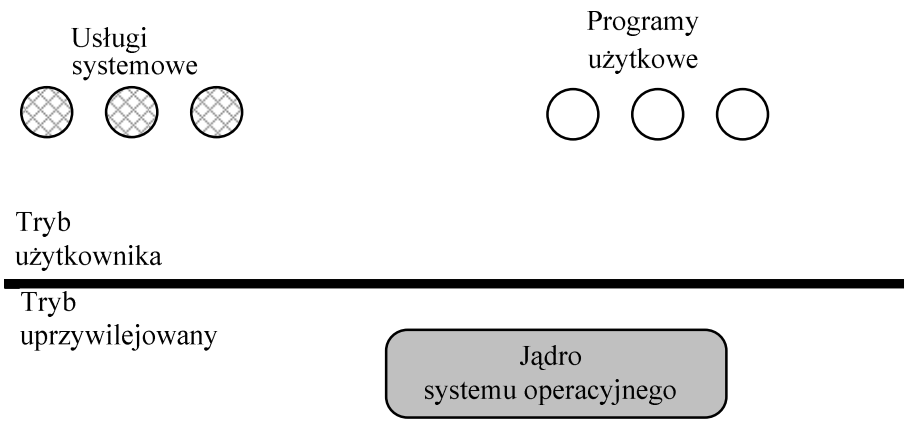
- programy użytkowe (ang. utility), takie jak: programy kompresji plików, archiwizacji danych itp.,
- programy narzędziowe – edytory znakowe i graficzne, kompilatory języków, konsolidatory (ang. linker), debugery itp.,
- programy usług dodatkowych – specjalne interfejsy użytkownika, kalkulator, gry itp.,
- biblioteki procedur.

Podział systemu operacyjnego na jądro i moduły pomocnicze umożliwia łatwe rozszerzanie systemu operacyjnego. W celu dodania nowej funkcji do systemu operacyjnego wystarczy opracować nową aplikację, co nie wymaga modyfikacji funkcji jądra.

3.2. Jądro w trybie uprzywilejowanym

W celu zapewnienia niezawodnego wykonywania programów użytkowych, system operacyjny, który także jest programem, powinien posiadać określone przywileje. W przeciwnym wypadku nieprawidłowo wykonujący się program użytkowy mógłby zakłócić pracę systemu operacyjnego komputera. Wszystkie wysiłki twórców systemu operacyjnego byłyby daremnymi, gdyby programy systemowe nie były chronione przed programami aplikacyjnymi. System operacyjny powinien posiadać wyjątkowe pełnomocnictwa po to, aby pełnić rolę arbitra w sytuacjach konfliktowych, mających miejsce podczas rywalizacji programów użytkowych o dostęp do zasobów komputera w środowisku wielozadaniowym.

Zapewnienie określonych przywilejów dla systemu operacyjnego jest możliwe wyłącznie dzięki specjalnym środkom sprzętowym. Procesor komputera powinien zapewniać wykonywanie programów w co najmniej dwóch trybach pracy – trybie użytkownika (ang. user mode) oraz trybie uprzywilejowanym, określanym także trybem jądra (ang. kernel mode). Programy wchodzące w skład systemu operacyjnego są wykonywane w trybie jądra, natomiast programy użytkowe – w trybie użytkownika (rys. 3.3).



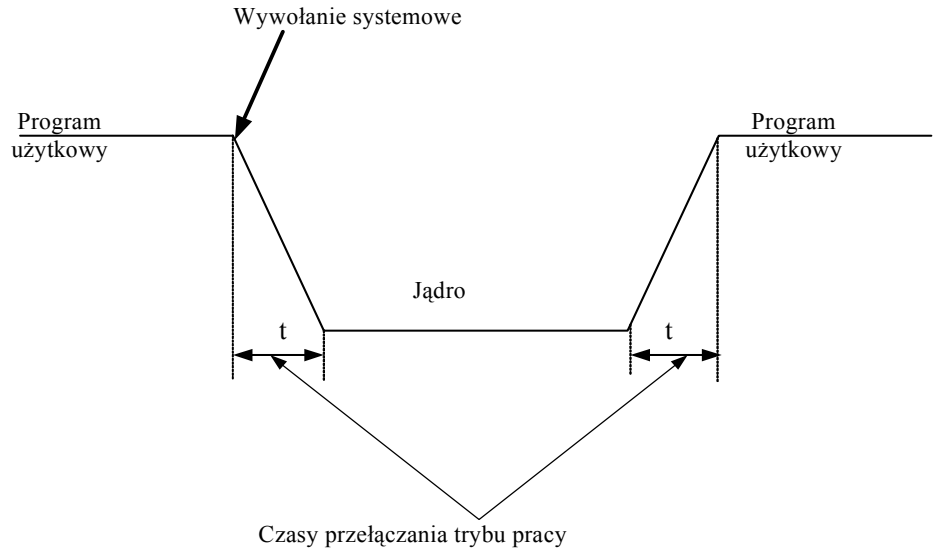
Rys. 3.3. Ogólna architektura systemu operacyjnego

Programy użytkowe pracujące w trybie użytkownika nie mogą wykonywać określonych rozkazów procesora związanych wyłącznie z trybem jądra, jak np. rozkazów związanych z przełączaniem kontekstu, operacjami WE/WY, ochroną pamięci itp.

Analogicznie realizowany jest uprzywilejowany dostęp systemu operacyjnego do pamięci operacyjnej. Programy wchodzące w skład systemu operacyjnego, wykonujące się w trybie jądra, posiadają dostęp do całej przestrzeni adresowej komputera. Natomiast przekroczenie przez program użytkowy granicy obszaru pamięci przydzielonego aplikacji powoduje wstrzymanie realizacji aplikacji. Dzięki temu każda aplikacja pracuje wyłącznie w przestrzeni adresowej przydzielonej przez system operacyjny danej aplikacji. W ten sposób realizowana jest ochrona obszarów pamięci zajętych przez system operacyjny przed programami użytkowymi, lecz także wzajemna ochrona obszarów pamięci zajętych przez poszczególne aplikacje przed innymi aplikacjami. Granice przestrzeni adresowych przydzielonych poszczególnym aplikacjom określa system operacyjny.

Nie ma ścisłego związku między liczbą poziomów uprzywilejowania realizowanych sprzętowo (przez procesor), a liczbą logicznych poziomów uprzywilejowania zaimplementowaną w danym systemie operacyjnym. Dla przykładu, na podstawie czterech sprzętowych poziomów uprzywilejowania procesora Pentium, system operacyjny OS/2 realizuje trzypoziomowy system uprzywilejowania, a systemy operacyjne UNIX i Windows NT – dwupoziomowy.

Zwiększenie stabilności systemu operacyjnego, uzyskane dzięki przełączaniu jądra w tryb uprzywilejowany, odbywa się kosztem pewnego zmniejszenia szybkości wywołań systemowych. Wywołanie przez program użytkowy funkcji wchodzącej w skład jądra systemu wymaga przełączenia procesora z trybu użytkownika do trybu uprzywilejowanego, a po zakończeniu wykonywania funkcji jądra – powrotu z trybu uprzywilejowanego w tryb użytkownika (rys. 3.4).

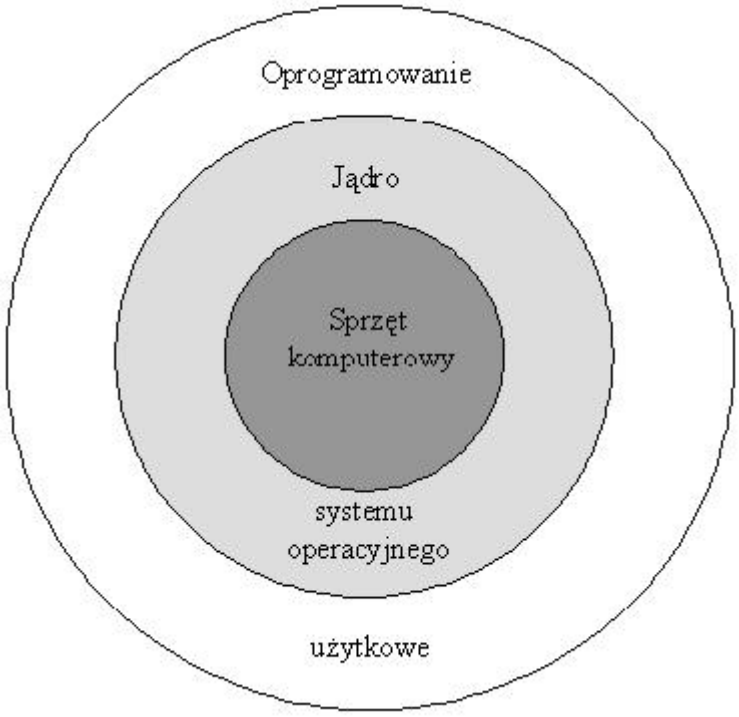


Rys. 3.4. Przełączanie trybu pracy podczas wywołania systemowego

Architektura systemu operacyjnego oparta na jądrze pracującym w trybie uprzywilejowanym oraz aplikacjach w trybie użytkownika jest stosowana we wszystkich współczesnych systemach operacyjnych (UNIX, OS/2, Windows NT). Jako wyjątek od przedstawionej reguły można podać system operacyjny NetWare, w którym realizacja programów jądra systemu operacyjnego, jak i określonych aplikacji (ładowanych poprzez moduł NML), odbywa się na jednym poziomie uprzywilejowania. Dzięki temu odwołania aplikacji do funkcji jądra są wykonywane szybciej, ale nie funkcjonuje sprzętowa ochrona pamięci, zajmowanej przez jądro, przed wadliwie funkcjonującymi aplikacjami.

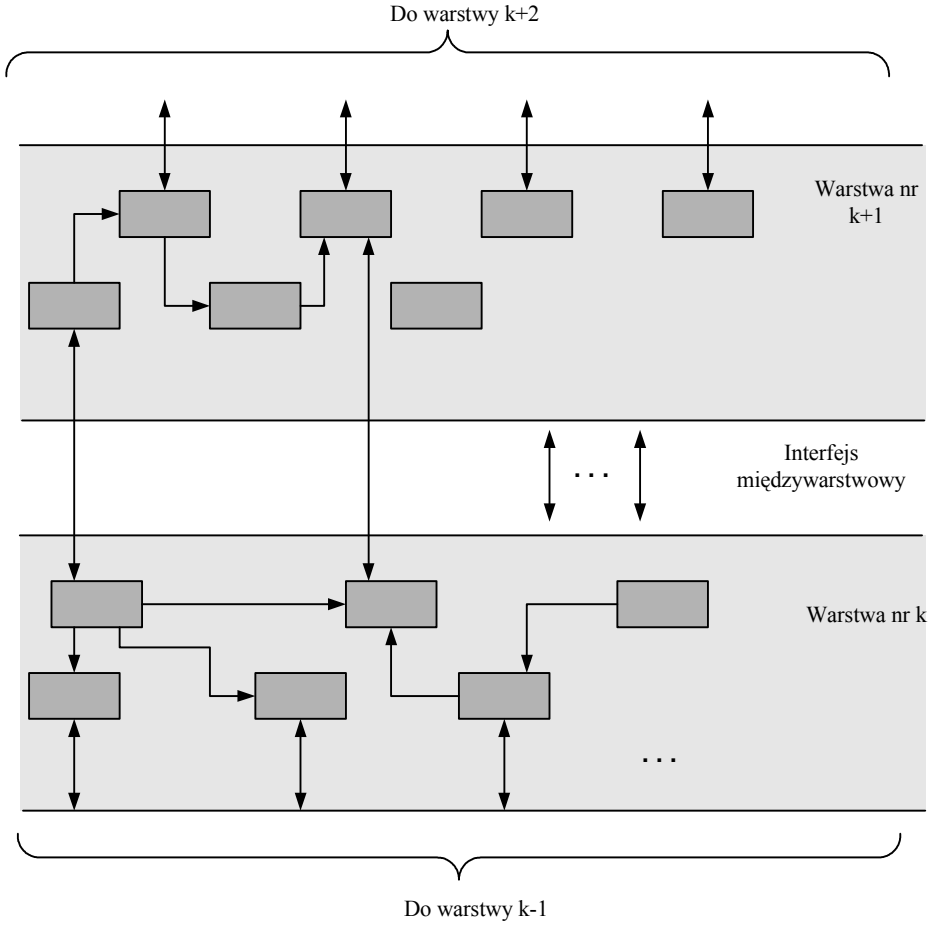
3.3. Wielowarstwowa struktura systemu operacyjnego

System komputerowy pracujący pod kontrolą systemu operacyjnego można traktować jako system trójwarstwowy: najniżej leżąca warstwa zawierająca sprzęt komputerowy, jądro systemu operacyjnego oraz najwyżej leżąca warstwa oprogramowania użytkowego (rys. 3.5). Zwykle warstwową strukturę systemu przedstawia się w postaci koncentrycznych okręgów, obrazując w ten sposób fakt, że dana warstwa może współpracować jedynie z warstwami sąsiednimi. Wynika z tego, że program użytkowy nie może bezpośrednio wykonywać operacji na sprzęcie, a wyłącznie za pośrednictwem systemu operacyjnego.



Rys. 3.5. Trójwarstwowa struktura systemu komputerowego

Podejście wielopoziomowe jest efektywną i uniwersalną metodą dekompozycji systemów złożonych dowolnego typu, w tym także systemów oprogramowania. Zgodnie z tym podejściem system składa się z szeregu warstw zorganizowanych w sposób hierarchiczny. Każda warstwa oprogramowania wykonuje szereg usług dla warstwy wyższej, a komunikacja między warstwami odbywa się za pomocą dobrze zdefiniowanego interfejsu (rys. 3.6). Realizacja usług w danej warstwie odbywa się poprzez wywołania usług warstwy niższej. Dzięki temu warstwy wyższe udostępniają usługi bardziej złożone, na których wykonanie składa się szereg funkcji warstw niższych. Wykonanie funkcji w danej warstwie może być realizowane przez pojedynczy moduł danej warstwy lub może być związane z wywołaniami innych modułów danej warstwy.



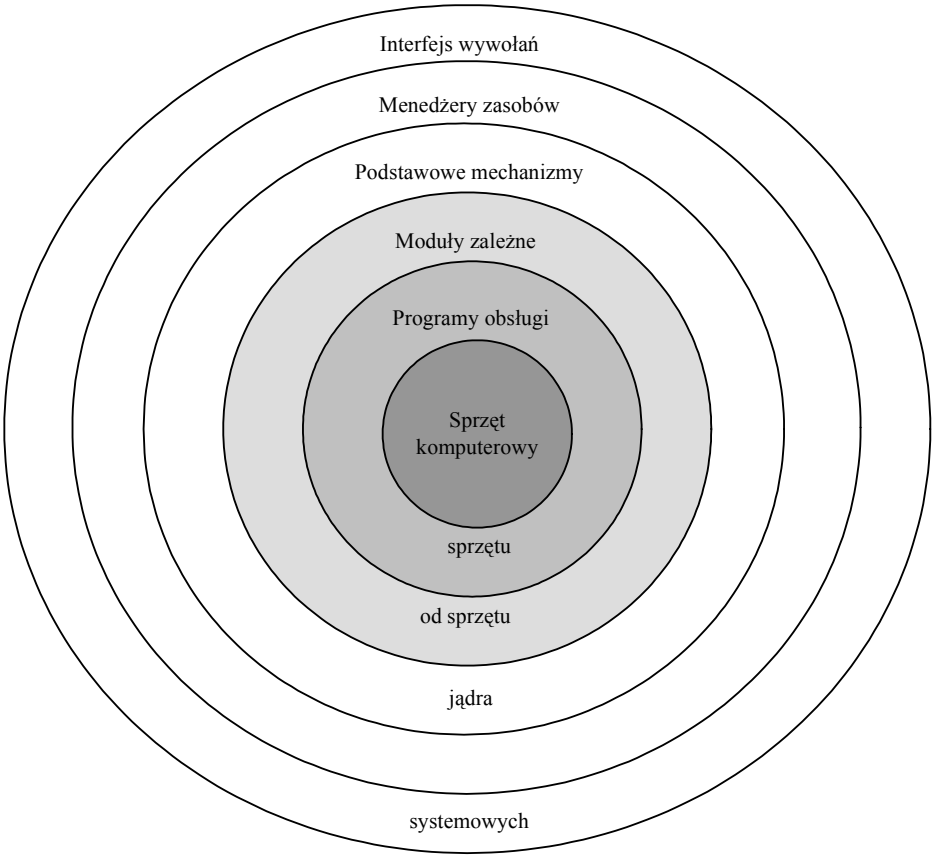
Rys. 3.6. Wielowarstwowa koncepcja budowy złożonego oprogramowania

W systemie operacyjnym jego jądro może być zbudowane z następujących warstw (rys. 3.7):

- *Środki sprzętowego wspomagania systemu operacyjnego.* Część funkcji systemu operacyjnego jest realizowana sprzętowo przez procesor. Do funkcji tych można zaliczyć np. wspomaganie trybu uprzywilejowanego, system przerwań, przełączanie kontekstu, ochronę obszarów pamięci itd.,
- *Komponenty systemu operacyjnego zależne od sprzętu.* W warstwie tej realizowane są funkcje ściśle związane z platformą sprzętową danego komputera. Zadaniem tej warstwy jest separacja wyżej leżących warstw jądra od szczegółów sprzętowych, co uniezależnia wyższe warstwy jądra systemu operacyjnego od platformy sprzętowej.
- *Podstawowe mechanizmy jądra.* W warstwie tej realizowane są elementarne czynności jądra systemu, takie jak programowe operacje związane z przełączaniem kontekstu, szeregowanie przerwań, przenoszenie stron między pamięcią i dyskiem itd.,
- *Menedżery zasobów.* W skład tej warstwy wchodzi szereg modułów realizujących strategiczne operacje związane z zarządzaniem podstawowymi zasobami systemu komputerowego. Zwykle w warstwie tej funkcjonują następujące menedżery: procesów (program szeregujący), urządzeń WE/WY, pamięci operacyjnej oraz systemu plików (często menedżer systemu plików wchodzi w skład menedżera urządzeń WE/WY),
- *Interfejs wywołań systemowych.* Warstwa ta tworzy interfejs API dla aplikacji. Funkcje API obsługujące wywołania systemowe udostępniają zasoby systemowe wykonywanym aplikacjom bez ujawniania szczegółów realizacyjnych. Np. w systemie UNIX systemowe wywołanie z programu napisanego w języku C: `fd=open(„/katalog/plik.txt”,O_RDWR)` otwiera plik do odczytu i zapisu, zwracając deskryptor pliku (liczba całkowita z przedziału 0..19). W celu wykonania tak złożonej operacji, wielokrotnie są wywoływane funkcje warstw jądra leżące w warstwach niższych.

3.4. Zależność oprogramowania systemowego od sprzętu oraz przenoszenie systemu operacyjnego

Wiele systemów operacyjnych funkcjonuje na różnych platformach sprzętowych bez konieczności dokonywania głębokich zmian w systemie. Można to wyjaśnić faktem, że wiele platform sprzętowych ma analogiczne cechy wspomagające funkcjonowanie systemów operacyjnych. Dzięki temu można wydzielić stosunkowo wąską warstwę komponentów jądra systemu operacyjnego zależnych od sprzętu, podczas gdy pozostałe warstwy jądra są niezależne od sprzętu.



Rys. 3.7. Wielowarstwowa struktura jądra systemu operacyjnego

3.4.1. Sprzętowe środki wspomagania systemu operacyjnego

Decyzję o tym, które funkcje systemu operacyjnego będą realizowane sprzętowo, a które programowo, są podejmowane podczas projektowania systemu operacyjnego. Nie istnieje ostra granica między funkcjami realizowanymi sprzętowo i programowo. Niemniej praktycznie wszystkie współczesne platformy sprzętowe są wyposażone w typowy zbiór układów wspomagania systemu operacyjnego, do których można zaliczyć układy:

- wspomagania trybu uprzywilejowanego,
- translacji adresów,
- przełączania procesów,
- system przerwań,
- zegar systemowy,
- ochrony obszarów pamięci.

Układ wspomagania trybu uprzywilejowanego jest oparty na rejestrze systemowym procesora, określanym mianem *słowa stanu procesora*. Rejestr ten zawiera szereg znaczników określających tryb pracy procesora, a w tym znacznik trybu uprzywilejowanego. Zmiana trybu pracy procesora odbywa się poprzez wykonanie odpowiedniego rozkazu zmieniającego słowo stanu procesora.

Układy translacji adresów wykonują operacje przekształcenia adresów wirtualnych, zawartych w wykonywanych programach, na adresy fizyczne. Do translacji adresów wykorzystywane są odpowiednie tablice przechowywane w pamięci operacyjnej, a w rejestrach procesora zapisane są jedynie wskaźniki (numery pozycji) na odpowiednie pozycje tablic. Środki translacji adresów sprzętowo realizują odpowiednie algorytmy przekształcania adresów, do których można zaliczyć *segmentację* oraz *stronicowanie* pamięci.

Układy przełączania procesów są przeznaczone do szybkiego zapisu w pamięci kontekstu procesu, którego realizacja zostaje wstrzymana, oraz odczytu z pamięci kontekstu procesu, który staje się aktywny. Kontekst procesu zawiera zwykle wszystkie rejestry ogólnego przeznaczenia procesora, rejestr flag oraz te rejestry systemowe, które są związane z konkretnym procesem a nie z systemem operacyjnym (np. wskaźnik na tablicę translacji adresów procesu). Przełączenie kontekstu odbywa się w wyniku realizacji określonego rozkazu procesora.

System przerwań pozwala komputerowi reagować na zdarzenia zewnętrzne, synchronizację wykonywania procesów, pracę układów WE/WY oraz przełączanie procesów. Wystąpienie przerwania informuje procesor o wystąpieniu w systemie komputerowym określonego zdarzenia niezsynchronizowanego z cyklem pracy procesora. Przykładami takich zdarzeń mogą być: zakończenie operacji WE/WY realizowanej przez urządzenie zewnętrzne, nieprawidłowe zakończenie operacji arytmetycznej (np. dzielenie przez zero), upływ określonego interwału czasu itp. Po wystąpieniu określonych warunków generacji przerwania źródło przerwania wystawia sygnał elektryczny informujący procesor o przerwaniu. Sygnał ten przerywa wykonywanie przez procesor aktualnego programu i powoduje przejście do realizacji wcześniej przygotowanej procedury obsługi przerwania. Z zasady operacja skoku do procedury obsługi przerwania jest związana ze zmianą poziomu uprzywilejowania. Po zakończeniu procedury obsługi przerwania następuje powrót do programu realizowanego przed wystąpieniem przerwania.

Zegar systemowy jest niezbędny w systemie operacyjnym do odmierzania interwałów czasowych. Jest to najczęściej licznik elektroniczny zliczający impulsy o określonej częstotliwości, poczynając od zadanej wartości początkowej do zera. Wyzerowanie licznika powoduje generację przerwania, które następnie jest obsługiwane przez odpowiednią procedurę obsługi. Zegar systemowy służy przede wszystkim do odmierzania czasu realizacji poszczególnych procesów w systemie operacyjnym z podziałem czasu.

Układy ochrony obszarów pamięci zapewniają na poziomie sprzętowym kontrolę adresów wystawianych przez procesor. Wystawiony adres porównywany jest z przestrzenią adresową przydzieloną danemu procesowi i w przypadku gdy adres przekracza zadany zakres, generowane jest przerwanie informujące system operacyjny o naruszeniu zadanych granic pamięci.

3.4.2. Komponenty systemu operacyjnego zależne od sprzętu

Jądro systemu operacyjnego może być zaprojektowane tak, że tylko część modułów jest zależna od sprzętu, podczas gdy pozostałe moduły są uniwersalne i nie zależą od platformy sprzętowej. Warstwa jądra systemu zależna od sprzętu jest bezpośrednio związana z warstwą środków sprzętowego wspomagania systemu operacyjnego. Taka lokalizacja znacznie ułatwia operację przeniesienia systemu operacyjnego na nową platformę sprzętową. Zasadniczym problemem występującym podczas przenoszenia systemu operacyjnego na inną platformę sprzętową jest inna lista rozkazów procesora. Problem ten jest rozwiązany dzięki tworzeniu oprogramowania systemowego w języku wysokiego poziomu (np. w języku C). W ten sposób, jeżeli producent komputera udostępnia kompilator danego języka wysokiego poziomu, to wystarczy oprogramowanie systemowe ponownie skompilować na nowej platformie sprzętowej.

Jednak w wielu przypadkach różnice w organizacji komputerów sięgają bardzo głęboko i przezwyciężenie ich jest bardzo trudne. Np. dla komputerów z jednym i z dwoma procesorami niezbędne są odmienne algorytmy przydziału procesora w systemie z podziałem czasu. Analogicznie w przypadku gdy procesor nie realizuje mechanizmów pamięci wirtualnych, moduł zarządzania pamięcią należy praktycznie napisać od nowa. W takich przypadkach niezbędne jest wprowadzenie do kodu systemu operacyjnego niezbędnych zmian związanych ze specyfiką platformy sprzętowej.

Szczególne miejsce wśród modułów jądra systemu operacyjnego posiadają niskopoziomowe sterowniki urządzeń zewnętrznych. Z jednej strony sterowniki te wchodzą w skład menedżera urządzeń WE/WY leżącego w wysoko-poziomowej warstwie jądra systemu operacyjnego (rys. 3.7), z drugiej strony niskopoziomowe sterowniki urządzeń uwzględniają wszystkie szczegóły urządzeń zewnętrznych, co powoduje, że można je przyporządkować do niskopoziomowych modułów zależnych od platformy sprzętowej. Taka dwoistość powoduje, że ścisła hierarchiczna budowa systemu operacyjnego nie jest możliwa.

3.4.3. Przenoszenie systemu operacyjnego na inną platformę

W przypadku gdy kod systemu operacyjnego może być stosunkowo łatwo przeniesiony na platformę sprzętową zawierającą inny typ procesora oraz posiadającą inną architekturę, to system taki określa się mianem mobilnego

(ang. portable). Aby system operacyjny był systemem mobilnym, należy spełnić następujące warunki.

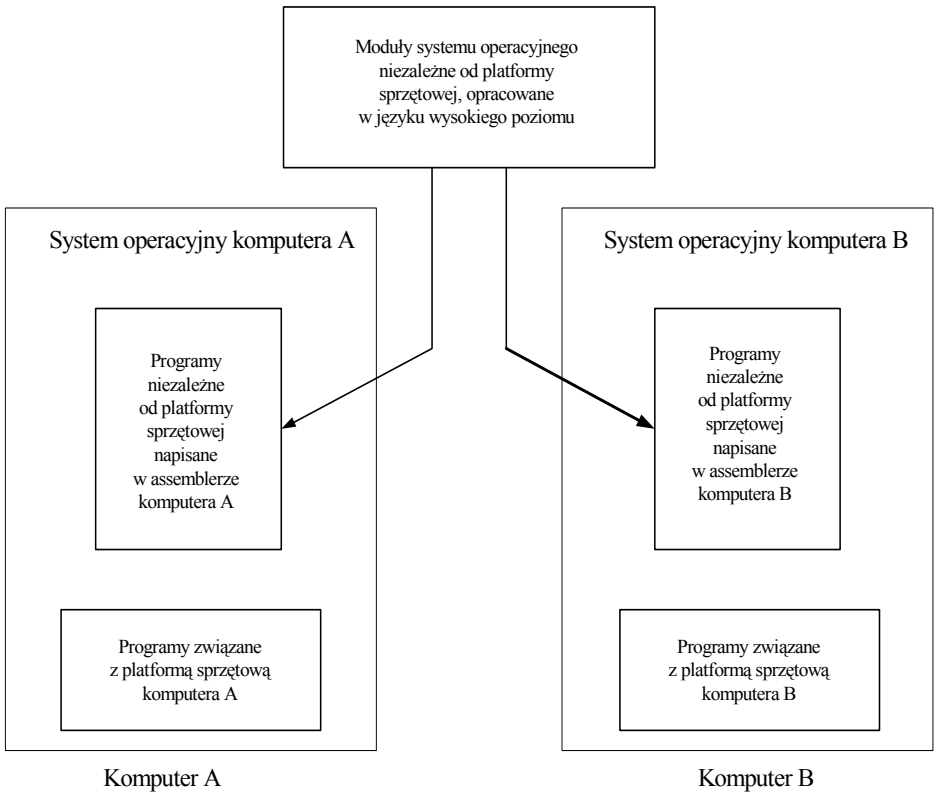
- Zdecydowana większość oprogramowania systemowego powinna być napisana w języku wysokiego poziomu, dla którego są dostępne kompilatory na wielu platformach sprzętowych (np. język C). W przypadku gdy oprogramowanie systemowe jest napisane w assemblerze, przeniesienie systemu operacyjnego będzie możliwe wyłącznie na komputery zawierające procesory charakteryzujące się tą samą listą rozkazów. Z zasady w assemblerze pisana jest wyłącznie część oprogramowania systemowego ściśle związana ze sprzętem.
- Rozmiar części kodu systemu operacyjnego, związanego z konkretną platformą sprzętową, powinien być zredukowany do minimum.
- Fragmenty kodów opracowane na konkretną platformę sprzętową powinny być izolowane od pozostałych programów i umieszczone w określonych modułach systemu operacyjnego.

W idealnym przypadku warstwa jądra zależna od platformy sprzętowej powinna w pełni izolować pozostałą część systemu operacyjnego od szczegółów związanych z architekturą komputera (pamięci cache, sterowniki przerwań itp.). W wyniku tego ma miejsce podmiana komputera rzeczywistego maszyną wirtualną, jednakową dla wszystkich platform sprzętowych, a wszystkie warstwy jądra systemu leżące powyżej warstwy komponentów zależnych od sprzętu, mogą być napisane dla tej maszyny wirtualnej. Na rys. 3.8 przedstawiono schematycznie operację instalacji danego systemu operacyjnego na różnych platformach sprzętowych.

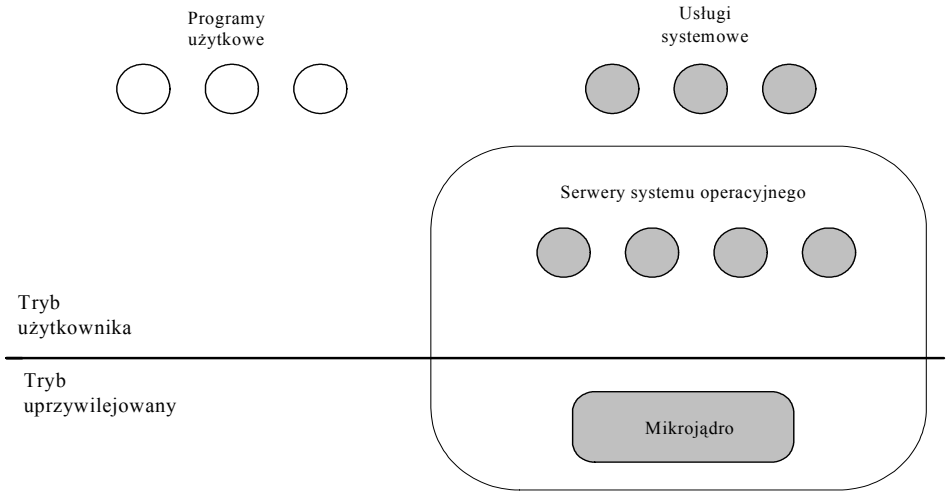
3.5. Systemy operacyjne z mikrojądrem

Architektura systemów operacyjnych z mikrojądrem jest architekturą alternatywną w stosunku do wcześniej analizowanej architektury klasycznej, określanej mianem architektury z monolitycznym jądrem. Cechą charakterystyczną takiego systemu operacyjnego jest fakt, że tylko bardzo niewielka część systemu, określana mianem mikrojądra, pracuje w trybie uprzywilejowanym (rys. 3.9). W skład mikrojądra z zasady wchodzą moduły zależne od platformy sprzętowej oraz moduły wykonujące podstawowe funkcje jądra związane z zarządzaniem procesami, obsługą przerwań, zarządzaniem pamięcią wirtualną, przesyłaniem komunikatów między procesami. Wszystkie pozostałe funkcje jądra systemu operacyjnego są realizowane jako programy użytkowe, wykonywane w trybie użytkownika.

Usługi jądra systemu operacyjnego, przeniesione do trybu użytkownika, są w tym przypadku realizowane w postaci serwerów. Podstawą pracy systemu operacyjnego z mikrojądrem jest więc architektura klient-serwer, w której programy użytkowe zgłaszają się do jądra o wykonanie określonej usługi, a jądro systemu zwraca się o wykonanie tej usługi do serwera danej usługi.



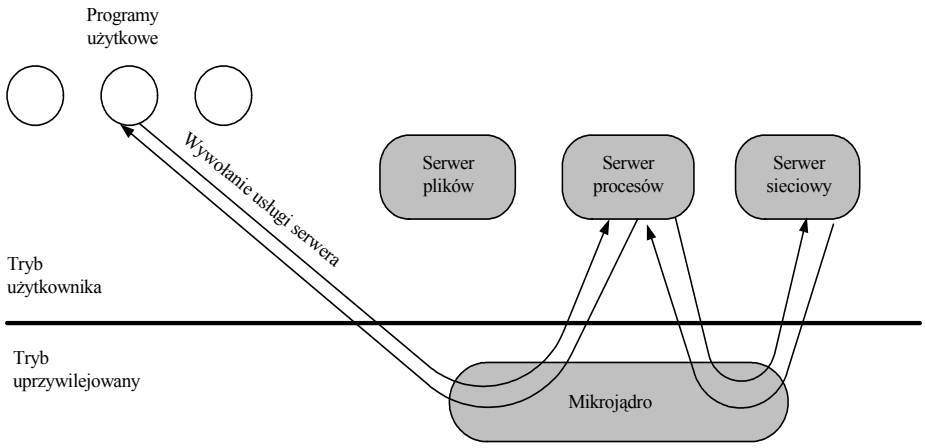
Rys. 3.8. Struktura oprogramowania systemowego z punktu widzenia platformy sprzętowej



Rys. 3.9. Architektura systemu operacyjnego z mikrojądrem

pracującego w trybie użytkownika. Wynika z tego, że w systemie operacyjnym z mikrojądrem musi funkcjonować efektywny mechanizm przekazywania komunikatów między procesami (jest to jedna z podstawowych funkcji jądra).

Na rys. 3.10 schematycznie przedstawiono mechanizm wywołania określonej usługi przez program użytkowy w systemie operacyjnym z mikrojądrem. Klient, którym może być program użytkowy lub inny komponent systemu operacyjnego, zwraca się do jądra systemu o wykonanie określonej funkcji realizowanej przez odpowiedni serwer (bezpośrednie odwołanie się do innego procesu – bez udziału jądra – jest w systemie niemożliwe). Jądro przekazuje komunikat, zawierający nazwę funkcji oraz parametry, do odpowiedniego serwera, a serwer realizuje odpowiednią usługę i wynik przekazuje do jądra, które z kolei przesyła odpowiedni komunikat do klienta.



Rys. 3.10. Realizacja wywołania systemowego dla systemu operacyjnego z mikrojądrem

Systemy operacyjne oparte na koncepcji mikrojądra w znacznym stopniu spełniają wymogi stawiane przed współczesnymi systemami operacyjnymi i charakteryzują się łatwością przenoszenia na inne platformy sprzętowe, łatwością rozbudowy, niezawodnością a także łatwością budowy systemów rozproszonych. Za te zalety trzeba zapłacić jednak cenę w postaci zmniejszenia wydajności systemu, co jest główną wadą systemów z mikrojądrem. Łatwość przenoszenia systemu na inne platformy sprzętowe wynika z faktu, że cały kod zależny od sprzętu jest zawarty w mikrojądrze. Dzięki temu dla nowej platformy sprzętowej należy opracować jedynie nowe mikrojądro, podczas gdy wszystkie usługi systemowe wystarczy skompilować na nowej platformie.

Łatwość rozbudowy systemów operacyjnych z mikrojądrem wynika przede wszystkim z faktu, że dodanie nowej funkcjonalności do systemu polega na

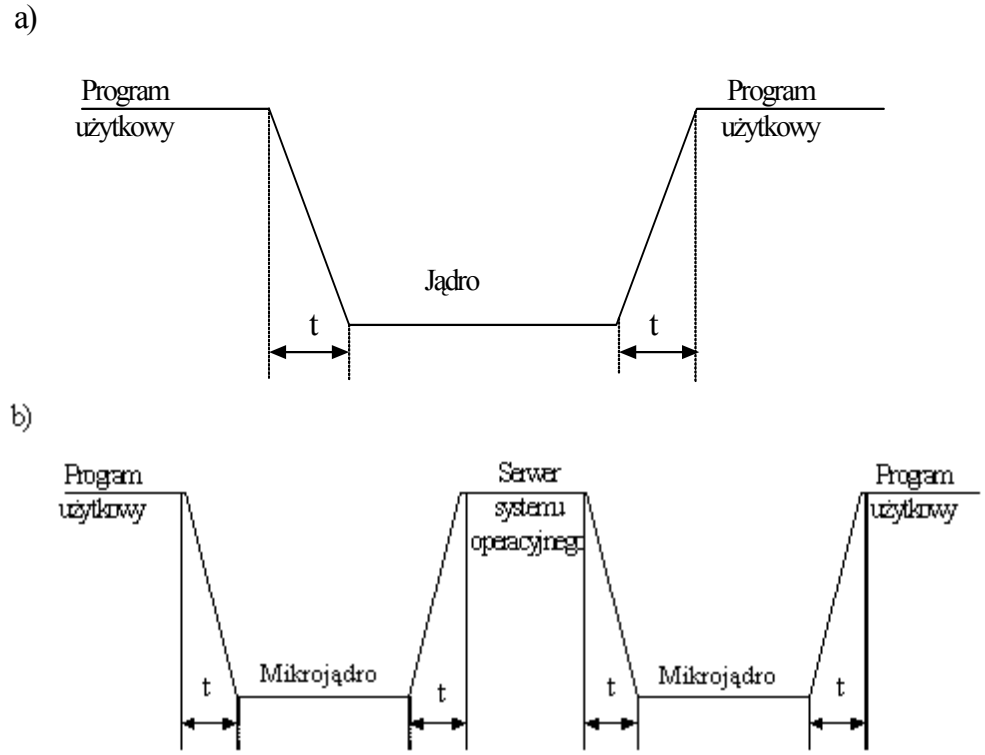
opracowaniu nowego programu użytkowego. Także operacja usunięcia określonej funkcjonalności systemu operacyjnego może być wykonana bardzo prosto (dzięki temu można przystosowywać architekturę konkretnej wersji systemu operacyjnego do potrzeb użytkownika).

Architektura z mikrojądrem zwiększa także niezawodność systemu operacyjnego. Każdy z serwerów jest wykonywany w postaci odrębnego procesu w wyodrębnionej przestrzeni adresowej, co zapewnia jego ochronę przed innymi serwerami systemu operacyjnego. W przypadku gdy określony serwer pracuje niewłaściwie, istnieje możliwość jego wstrzymania i ponownego uruchomienia bez konieczności restartu całego systemu operacyjnego. Poza tym skoro serwery są wykonywane w trybie użytkownika, nie posiadają bezpośredniego dostępu do sprzętu i nie mogą modyfikować pamięci, w której przechowywane jest mikrojądro. Sytuacja taka nie ma miejsca w systemach operacyjnych z monolitycznym jądrem, gdzie wszystkie moduły jądra mogą wpływać wzajemnie na siebie.

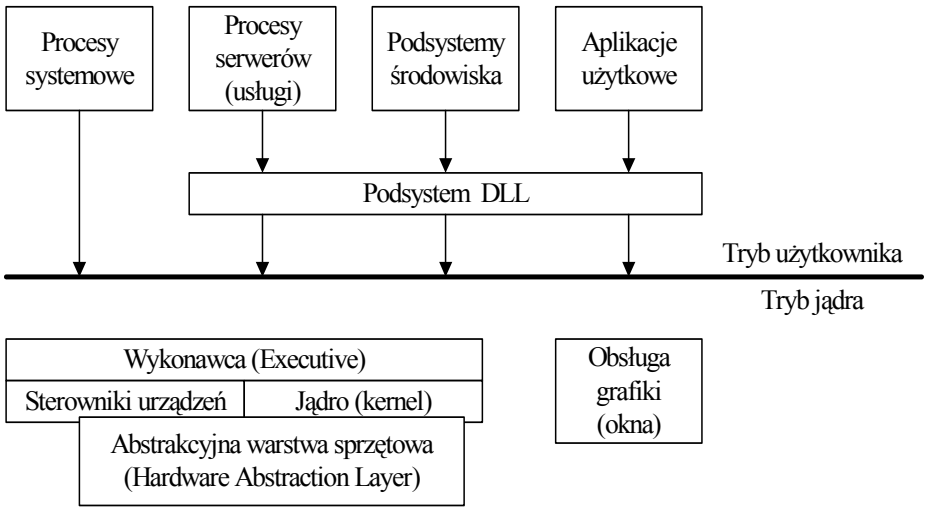
Model systemu operacyjnego z mikrojądrem jest szczególnie korzystny dla realizacji systemów rozproszonych, ponieważ wykorzystuje mechanizmy przekazywania komunikatów, analogiczne do mechanizmów sieciowych – współdziałanie klientów i serwerów poprzez wymianę komunikatów. Serwery systemu operacyjnego mogą być wykonywane tak na jednym, jak i wielu różnych komputerach połączonych siecią. Dzięki temu mikrojądro danego komputera może zlecić wykonanie usługi lokalnemu serwerowi lub serwerowi wykonywanemu na innym komputerze w sieci (komunikaty mogą być przesyłane lokalnie w ramach jednego węzła sieci lub sieciowo między różnymi węzłami sieci).

Niestety architektura systemu operacyjnego z mikrojądrem charakteryzuje się mniejszą wydajnością. Wynika to przede wszystkim z większej liczby przełączeń procesora z trybu użytkownika do trybu jądra i odwrotnie. W systemie operacyjnym z monolitycznym jądrem (rys. 3.11a) wykonanie wywołania systemowego jest związane z dwoma przełączeniami trybów pracy procesora, a w systemie z mikrojądrem – minimum czterema (rys. 3.11b).

Współczesne systemy operacyjne próbują łączyć cechy systemów operacyjnych z monolitycznym jądrem i systemów z mikrojądrem. Przykładem takiego systemu operacyjnego jest system Windows NT. W systemie tym komponenty mające duży wpływ na jego wydajność pracują w trybie jądra, w którym mogą współpracować ze sprzętem oraz innymi komponentami bez konieczności przełączania kontekstu i trybu pracy. Np. menedżery: pamięci, cache, obiektów, bezpieczeństwa, protokołów sieciowych, systemu plików, procesów i wątków pracują w trybie jądra. Komponenty te są dobrze chronione przed aplikacjami ponieważ nie mają dostępu do kodu i danych w trybie uprzywilejowanym (jądra). Uproszczona architektura systemu Windows NT jest przedstawiona na rys. 3.12.



Rys. 3.11. Przełączanie trybów pracy podczas wywołań systemowych
a) system operacyjny z monolitycznym jądrem
b) system operacyjny z mikrojądrem



Rys. 3.12. Uproszczona architektura systemu Windows NT

Do podstawowych składowych systemu Windows wykonywanych w trybie użytkownika należy zaliczyć:

- specjalne procesy systemowe, takie jak proces logowania, czy menedżer sesji, które nie są usługami Windows NT (nie są uruchamiane przez sterownik usług),
- procesy serwerów będące usługami Windows NT, np. Event Log,
- podsystemy środowiska, które udostępniają usługi systemu Windows NT aplikacjom użytkowym. Windows NT dostarcza trzy podsystemy środowiska: Win32, POSIX, OS/2,
- aplikacje użytkowe. Może być pięć typów aplikacji: Win32, Windows 3.1, MS-DOS, POSIX, OS/2.

Aplikacje użytkowe nie wywołują bezpośrednio usług systemu Windows NT, lecz pośrednio poprzez jedną lub więcej bibliotek dynamicznych – Dynamic Link Libraries (DLL). Rola podsystemu DLL polega na tłumaczeniu dokumentowanych funkcji interfejsu API na odpowiednie nieudokumentowane wywołania usług systemowych Windows NT.

Część systemu operacyjnego pracującego w trybie jądra zawiera następujące komponenty:

- wykonawca (executive) WINDOWS NT zawiera podstawowe usługi systemowe, takie jak zarządzanie: pamięcią, procesami i wątkami, bezpieczeństwem, podsystemem WE/WY; mechanizmy komunikacji międzyprocesowej (funkcje te wywoływane są przez podsystem Win32 API lub inny podsystem środowiska),
- jądro systemu wykonuje niskopoziomowe operacje, takie jak szeregowanie wątków, obsługa przerwań i wyjątków, synchronizacja wieloprocesorowa,
- abstrakcyjna warstwa sprzętowa (ang. HAL – Hardware Abstraction Layer) zawiera programy, które izolują jądro, sterowniki urządzeń oraz pozostałe programy konkretnej platformy sprzętowej,
- sterowniki urządzeń zawierają programy będące sterownikami systemu plików oraz innych urządzeń, które tłumaczą wywołania WE/WY na konkretne żądania obsługi urządzeń,
- podsystem obsługi grafiki obsługuje graficzny interfejs użytkownika.

3.6. Kompatybilność i różnorodne środowiska użytkowe

Można wyróżnić dwa rodzaje kompatybilności oprogramowania: kompatybilność na poziomie binarnym oraz kompatybilność na poziomie tekstów źródłowych. Z kompatybilnością binarną mamy do czynienia w przypadku, gdy program binarny (skompilowany, zawierający obraz procesu w pamięci operacyjnej) może być wykonany w środowisku innego systemu operacyjnego. Natomiast kompatybilność na poziomie tekstów źródłowych jest związana z możliwością przeniesienia źródłowego tekstu programu (najczęściej

w języku wysokiego poziomu) i skompilowania w innym systemie operacyjnym, po czym program może być wykonany w innym systemie operacyjnym. Dla zapewnienia kompatybilności na poziomie tekstów źródłowych, w obydwu systemach operacyjnych musi istnieć odpowiedni kompilator, a także niezbędna jest kompatybilność bibliotek oraz wywołań systemowych.

Kompatybilność na poziomie tekstów źródłowych jest ważna dla twórców programów aplikacyjnych, gdyż dysponują tekstami programów. Natomiast dla użytkownika końcowego, korzystającego z programów wykonywalnych, istotna jest kompatybilność binarna. Np. dla użytkownika, który zakupił pakiet Lotus 1-2-3 dla MS-DOS, jest istotne, aby mógł ten program uruchomić także w systemie Windows NT.

Dla kompatybilności binarnej, poza zgodnością procesorów oraz zgodnością zakresów adresów w pamięci operacyjnej, niezbędne jest spełnienie następujących warunków:

- zgodność wywołań funkcji API zawartych w programie,
- zgodność wewnętrznej struktury plików wykonywalnych.

Osiągnięcie kompatybilności binarnej, w przypadku gdy systemy komputerowe posiadają procesory o różnych architekturach, jest znacznie trudniejsze. W takich przypadkach są stosowane tzw. emulatory kodu binarnego. Emulator pobiera kolejne rozkazy programu binarnego przeznaczonego dla innego procesora i realizuje je programowo, z zastosowaniem procesora danego komputera. Wadą takiego rozwiązania jest jednak znaczne spowolnienie wykonywania programu.

3.6.1. Translacja bibliotek

Innym rozwiązaniem stosowanym w systemach operacyjnych jest wykorzystanie tzw. środowisk programów użytkowych. Jedną ze składowych, tworzącą środowisko programów użytkowych, jest zbiór funkcji interfejsu API udostępnianych przez system operacyjny dla programów użytkowych. W celu przyspieszenia realizacji „obcych" programów, zamiast wykonywać je rozkaz po rozkazie, jak ma to miejsce w przypadku emulatorów kodu binarnego, środowiska użytkowe symulują odwołania do funkcji bibliotecznych realizujących operacje analogiczne do funkcji interfejsu API. Oznacza to, że emulator kodu, w przypadku napotkania wywołania funkcji interfejsu API, wywołuje wcześniej przygotowaną i skompilowaną funkcję biblioteczną. Dzięki temu znaczna część kodu związana z realizacją funkcji interfejsu API jest wykonywana z pełną szybkością procesora danego komputera. Efektywność takiego podejścia wynika z faktu, że w systemach operacyjnych (w szczególności wykorzystujących graficzny interfejs użytkownika – GUI) realizacja funkcji interfejsu API wymaga 60-80% czasu procesora.

Aby program napisany dla jednego systemu operacyjnego mógł być wykonany w innym systemie nie wystarczy zapewnić zgodności interfejsów

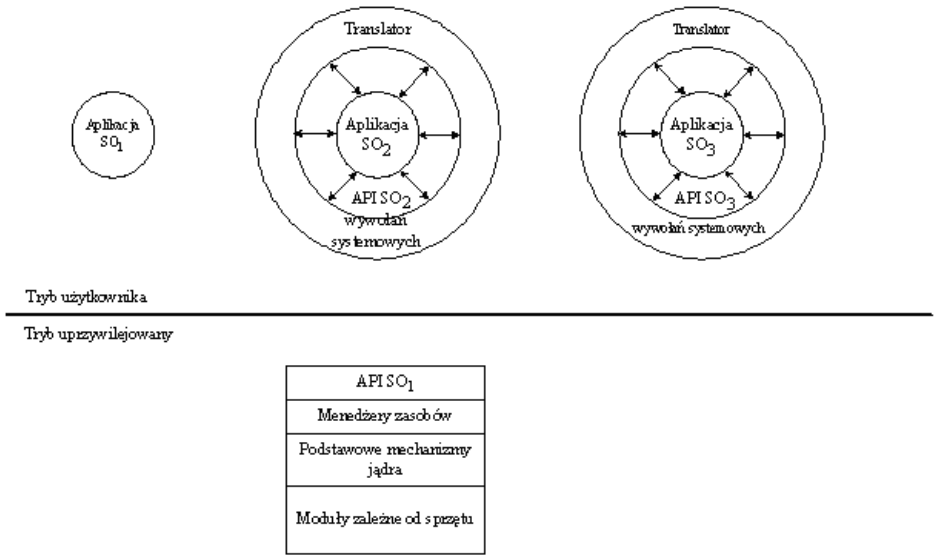
API. Może mianowicie być tak, że zasady obowiązujące w obydwu systemach operacyjnych są wzajemnie sprzeczne i nie da się ich pogodzić. Ma to np. miejsce w przypadku, gdy w jednym systemie operacyjnym aplikacja kliencka może wykonywać bezpośrednie operacje na urządzeniach WE/WY, a w drugim jest to zabronione. Z taką sytuacją spotykamy się w przypadku systemów operacyjnych MS-DOS i Windows NT. Dzięki wprowadzeniu trybu wirtualnych zadań procesora 8086, podczas pracy w trybie chronionym procesora PENTIUM (system Windows NT) jest możliwe wykonywanie programów przeznaczonych dla systemu operacyjnego DOS, włącznie z wywołaniami usług BIOS-u oraz funkcji DOS-u. Jeżeli jednak program napisany dla DOS-u, a wykonywany w środowisku Windows NT, wykonuje bezpośrednie operacje na sprzęcie (np. bezpośrednie operacje na rejestrach portu szeregowego czy operacje związane z obsługą przerwań), to system ochrony Windows NT wstrzymuje realizację takiego programu.

3.6.2. Realizacja środowisk programów użytkowych

Zadanie stworzenia środowiska użytkowego w pełni zgodnego ze środowiskiem innego systemu operacyjnego jest zadaniem złożonym, ściśle związanym ze strukturą systemu operacyjnego. Istnieją różne warianty budowy środowisk użytkowych, różniące się szczegółami architektury oraz możliwościami funkcjonalnymi, zapewniającymi różny stopień przenoszenia aplikacji. W wielu wersjach systemu operacyjnego UNIX translator środowiska użytkowego jest zrealizowany w postaci zwykłej aplikacji, natomiast w systemach operacyjnych z mikrojądrem (np. Windows NT), środowiska użytkowe są realizowane w postaci serwerów, wykonywanych w trybie użytkownika.

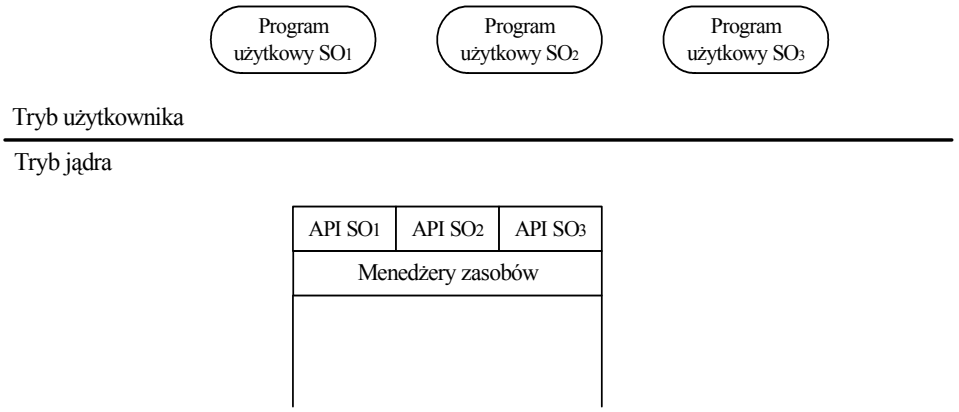
Jednym z najbardziej oczywistych wariantów realizacji środowisk użytkowych jest rozwiązanie oparte na standardowej wielowarstwowej architekturze systemu operacyjnego. Na rys. 3.13 system operacyjny SO_1 pozwala na wykonywanie nie tylko „swoich własnych" aplikacji, lecz także aplikacji przeznaczonych dla systemów operacyjnych SO_2 oraz SO_3. W tym celu w skład systemu operacyjnego SO_1 wchodzą specjalne aplikacje, określane mianem środowisk programów użytkowych, których zadaniem jest tłumaczenie interfejsów programowych innych systemów operacyjnych API SO_2 oraz API SO_3 na interfejs własnego systemu operacyjnego API SO_1. Np. jeśli systemem SO_1 będzie UNIX, a SO_2 – OS/2, to wywołanie systemowe DosExecPgm(), powodujące powołanie do życia nowego procesu w OS/2, powinno być przetłumaczone przez środowisko programów użytkowych na wywołanie fork(), typowe dla systemu UNIX.

Niestety, zbiory funkcji systemowych poszczególnych systemów operacyjnych różnią się znacznie. Ponadto realizacja analogicznych funkcji systemowych jest w poszczególnych systemach odmienna. Powoduje to, że realizacja takiego podejścia jest bardzo trudna.



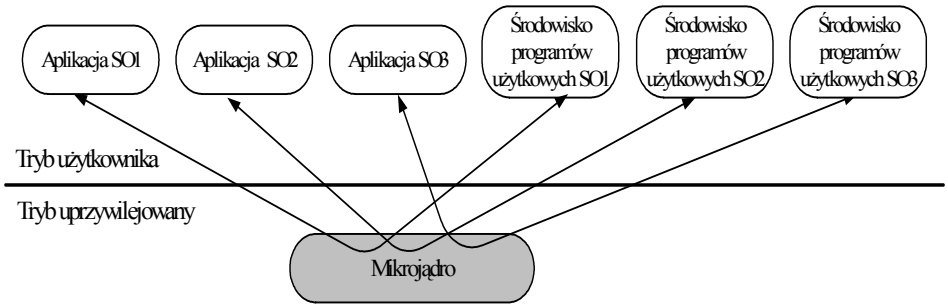
Rys. 3.13. Środowiska programów użytkowych tłumaczące wywołania systemowe

W drugim wariancie realizacji różnorodnych środowisk programów użytkowych system operacyjny jest wyposażony w kilka równoprawnych interfejsów programów użytkowych. W przykładzie przedstawionym na rys. 3.14 system operacyjny obsługuje aplikacje napisane dla systemów operacyjnych SO_1, SO_2 oraz SO_3. W tym celu w jądrze systemu operacyjnego umieszczono interfejsy programów użytkowych API SO_1, API SO_2 oraz API SO_3.



Rys. 3.14. Interfejsy programów użytkowych różnych systemów operacyjnych umieszczone w jądrze systemu

W inny sposób są realizowane środowiska programów użytkowych w systemach operacyjnych z mikrojądrem. W takim przypadku oddziela się wszystkie bazowe funkcje jądra, wspólne dla wszystkich środowisk, od mechanizmów systemowych specyficznych dla poszczególnych systemów operacyjnych. Dla nich są w tym przypadku opracowane serwery realizujące środowiska programów użytkowych (rys. 3.15).



Rys. 3.15. Realizacja środowisk programów użytkowych w systemie operacyjnym z mikrojądrem

Rozdział IV
PROCESY I WĄTKI

Jedną z najważniejszych funkcji systemu operacyjnego jest organizacja racjonalnego wykorzystania wszystkich, sprzętowych oraz informacyjnych, zasobów komputera. Do podstawowych zasobów mogą być zaliczone: procesory, pamięć, urządzenia zewnętrzne oraz dane i programy. Co prawda zadanie zarządzania zasobami było realizowane już w jednozadaniowych systemach komputerowych, to jednak główne trudności napotkano w systemach wielozadaniowych, w których szereg równocześnie realizowanych programów konkuruje o dostęp do zasobów komputera. Z tego powodu znaczna część tego rozdziału dotyczy wielozadaniowych systemów operacyjnych.

4.1. Wielozadaniowość

Wielozadaniowość (ang. multitasking) jest sposobem organizacji procesu obliczeniowego, w którym z wykorzystaniem jednego procesora naprzemiennie (z podziałem czasu) wykonuje się wiele programów jednocześnie. Programy te jednocześnie wykorzystują nie tylko procesor, lecz także inne zasoby komputera, jak pamięć operacyjną, pamięci masowe, urządzenia WE/WY, dane zapisane w plikach itd. Wielozadaniowość wprowadzono w celu zwiększenia efektywności wykorzystania systemu komputerowego.

Najczęściej stosowane są następujące kryteria efektywności systemów komputerowych:

- *przepustowość* – liczba zadań wykonanych w systemie w jednostce czasu,
- *wygoda pracy użytkownika* – związana z możliwością interaktywnej pracy z wieloma jednocześnie wykonywanymi programami,
- *reakcyjność* – cecha systemu operacyjnego związana z realizacją danego programu w zadanym odcinku czasu.

W zależności od wybranego kryterium efektywności, systemy operacyjne można podzielić na systemy przetwarzania wsadowego, systemy z podziałem czasu oraz systemy czasu rzeczywistego. Każdy typ systemu operacyjnego charakteryzuje się specyficznymi mechanizmami wewnętrznymi oraz odrębnymi obszarami zastosowań.

4.1.1. Systemy wielozadaniowe przetwarzania wsadowego

W celu zwiększenia przepustowości systemu komputerowego należy zminimalizować przestoje wszystkich urządzeń komputera, a w pierwszej kolejności – procesora. Przestoje procesora wynikają przede wszystkim z braku danych do przetwarzania (np. dane należy pobrać z pamięci masowej lub bezpośrednio od użytkownika za pośrednictwem terminala). Rozwiązaniem

problemu prowadzącym do zwiększenia efektywności wykorzystania procesora jest w takim przypadku przełączenie procesora na realizację innego zadania, w którym dane są gotowe do przetwarzania. Koncepcja ta leży u podstaw systemów przetwarzania wsadowego. Systemy przetwarzania wsadowego są w głównej mierze przeznaczone do rozwiązywania zadań obliczeniowych, w których czas odpowiedzi dla danego zadania może być dłuższy. Natomiast głównym kryterium efektywności systemu komputerowego jest maksymalna przepustowość liczona liczbą zadań rozwiązanych w jednostce czasu.

Dla systemu przetwarzania wsadowego można przedstawić następujący schemat funkcjonowania: w pierwszej kolejności jest tworzony pakiet zadań do wykonania, w którym każde zadanie zawiera zapotrzebowanie na zasoby systemowe, a następnie na podstawie pakietu tworzony jest zbiór zadań wykonywanych jednocześnie. Do jednoczesnego wykonania wybierane są zadania wymagające dostępu do różnych zasobów, aby zrównoważyć obciążenie wszystkich urządzeń systemu komputerowego.

Rozpatrzmy szczegółowo problem jednoczesnego wykonywania operacji WE/WY oraz operacji obliczeniowych. Z punktu widzenia architektur systemów komputerowych można wyróżnić dwa następujące przypadki:

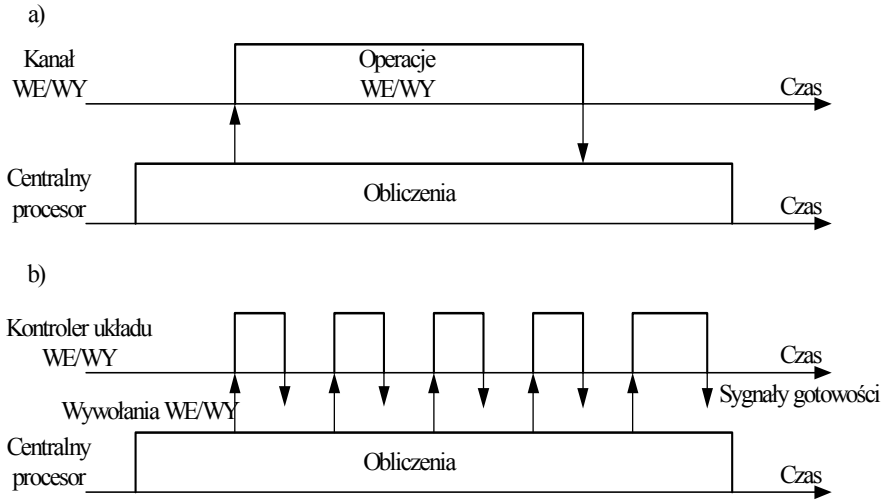
- komputer jest wyposażony w odrębny (jeden lub wiele) specjalizowany procesor WE/WY; w komputerach klasy mainframe procesory takie określane są mianem kanałów WE/WY,
- obsługa urządzeń WE/WY jest wykonywana przy pomocy wyspecjalizowanych sterowników (kontrolerów).

W pierwszym przypadku kanał posiada własny zbiór rozkazów służących do sterowania urządzeniami zewnętrznymi, a programy przeznaczone dla kanałów WE/WY mogą być przechowywane w pamięci operacyjnej komputera. W zbiorze rozkazów procesora centralnego występuje specjalny rozkaz do przekazania kanałowi parametrów oraz wskazania, który program WE/WY ma być wykonany. Po wykonaniu tego rozkazu procesor centralny oraz kanał WE/WY pracują równolegle (rys. 4.1a), a każdy realizuje własny program.

W drugim przypadku każde urządzenie WE/WY (lub grupa urządzeń tego samego typu) posiada swój własny kontroler, autonomicznie wykonujący pojedyncze rozkazy przekazywane z procesora centralnego, przy czym procesor centralny oraz kontroler pracują asynchronicznie. Ponieważ realizacja rozkazu przez kontroler WE/WY jest znacznie wolniejsza niż realizacja programu przez procesor centralny (urządzenia elektromechaniczne), podczas realizacji jednego rozkazu WE/WY procesor centralny może wykonać znaczną porcję obliczeń (rys. 4.1b). Poinformowanie procesora centralnego o zakończeniu realizacji rozkazu WE/WY może być wykonane z zastosowaniem przerwania lub poprzez cykliczne przepytywanie kontrolera przez procesor centralny (ang. pooling).

Podczas wykonywania jednego zadania stopień przyspieszenia wynikający z zastosowania wielozadaniowego systemu operacyjnego zależy przede wszystkim od realizowanego programu. Gdy w programie wykonywane są wyłącznie operacje obliczeniowe lub wyłącznie operacje WE/WY, to

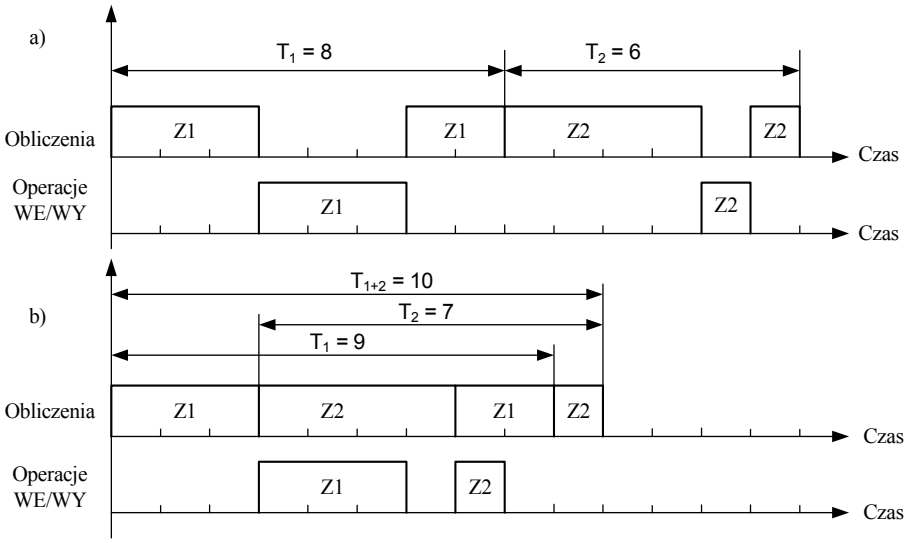
przyspieszenie praktycznie nie występuje. Ponadto, gdy realizowany program wymaga w pierwszej kolejności wczytania wszystkich danych, a dopiero w dalszej wykonuje obliczenia, przyspieszenie obliczeń także będzie niezauważalne.



Rys. 4.1. Równoległe wykonywanie obliczeń i operacji WE/WY
a) kanał WE/WY b) kontroler urządzenia WE/WY

W przypadku gdy w systemie wykonywanych jest wiele zadań jednocześnie, pojawia się możliwość równoległego wykonywania obliczeń w jednym zadaniu i operacji WE/WY w innym. Podczas gdy jedno zadanie oczekuje na jakieś zdarzenie, procesor może wykonywać inne zadanie. Ogólny czas realizacji zbioru zadań jest często znacznie krótszy od sumy czasów sekwencyjnego wykonania poszczególnych zadań (rys. 4.2a), jednak czas wykonania pojedynczego zadania w systemie wielozadaniowym może okazać się dłuższy od czasu realizacji tego zadania w przypadku gdyby było realizowane wyłącznie to zadanie. Wynika to z faktu, że mogą wystąpić sytuacje, w których proces po zakończeniu operacji WE/WY oczekuje na procesor, który wykonuje operacje w innym zadaniu (rys. 4.2b).

W wielozadaniowych systemach operacyjnych przetwarzania wsadowego przełączenie procesora z jednego zadania na drugie odbywa się na żądanie aktywnego zadania (gdy w zadaniu rozpoczyna się wykonywanie operacji WE/WY). Z tego powodu istnieje niebezpieczeństwo sytuacji, w której jedno zadanie zawładnie procesorem na długi czas, w przypadku gdy nie występują w nim operacje WE/WY. Może to znacznie wydłużyć realizację innych zadań, co zmniejsza efektywność pracy użytkownika.



Rys. 4.2. Realizacja dwóch zadań: a) w systemie jednozadaniowym, b) w systemie wielozadaniowym

4.1.2. Systemy wielozadaniowe z podziałem czasu

W systemach operacyjnych z podziałem czasu użytkownicy (lub jeden użytkownik) posiadają możliwość interaktywnej, jednoczesnej pracy z wieloma aplikacjami. Aby było to możliwe każda aplikacja musi w sposób cykliczny komunikować się z użytkownikiem (możliwość taka nie istniała w systemach przetwarzania wsadowego). System operacyjny wykonuje to poprzez periodyczne przełączanie aplikacji, nie czekając aż one same zwolnią procesor (w wyniku operacji WE/WY). System operacyjny naprzemiennie przydziela określony kwant czasu procesora każdej aplikacji, dzięki czemu użytkownicy posiadają możliwość ciągłego komunikowania się z aplikacją. Jeśli kwant czasu jest odpowiednio krótki, wszyscy użytkownicy pracujący jednocześnie na danym komputerze mają wrażenie, że posiadają komputer do wyłącznego użytkowania.

System operacyjny z podziałem czasu charakteryzuje się mniejszą przepustowością niż system przetwarzania wsadowego, gdyż współbieżnie realizowane są wszystkie uruchomione aplikacje, a nie aplikacje wybrane przez system operacyjny. Ponadto część mocy obliczeniowej systemu komputerowego musi być przeznaczona na operacje przełączania procesora między aplikacjami. Główną cechą systemów operacyjnych z podziałem czasu jest zwiększenie wygody pracy użytkownika kosztem pewnego zmniejszenia przepustowości systemu.

4.1.3. Systemy czasu rzeczywistego

W systemach operacyjnych komputerów używanych do sterowania procesami technicznymi stosowana jest jeszcze inna odmiana wielozadaniowości. W takim przypadku system komputerowy jest połączony z obiektem poprzez zestaw odpowiednich interfejsów, a jego głównym zadaniem jest wypracowywanie sygnałów sterujących obiektem. Procesy mające miejsce w obiekcie zachodzą z szybkością charakterystyczną dla zjawisk fizycznych (chemicznych, biologicznych itp.) występujących w obiekcie. Z tego powodu dla systemu komputerowego sterującego danym obiektem jest określony dopuszczalny czas, w ciągu którego musi być wykonany program realizujący zadaną operację. W przeciwnym wypadku w obiekcie może wystąpić awaria. Z tego powodu głównym kryterium efektywności systemu operacyjnego czasu rzeczywistego jest możliwość określenia interwałów czasowych (czasów reakcji systemu) między uruchomieniem poszczególnych programów a uzyskaniem wyników (np. sygnałów oddziaływania sterującego). Wymagania dotyczące czasu reakcji systemu zależą od specyfiki obiektu sterowanego (np. w przypadku robota przemysłowego czas ten może wynosić ok. 1 ms, a w przypadku bloku energetycznego – 1 s).

W systemach czasu rzeczywistego określa się zbiór programów, które mają być wykonywane współbieżnie (analogicznie jak w systemach z podziałem czasu), a wybór programów do wykonania odbywa się w reakcji na przerwania generowane na podstawie aktualnego stanu obiektu lub na podstawie opracowanego wcześniej harmonogramu. Zdolność sprzętu komputerowego oraz systemu operacyjnego do szybkiej odpowiedzi zależy przede wszystkim od czasu przełączania z jednego zadania na drugie oraz od czasu reakcji na sygnał przerwania. Jeśli np. po wystąpieniu przerwania procesor musi sprawdzić setki potencjalnych źródeł przerwań, to reakcja systemu może być zbyt wolna.

W systemach czasu rzeczywistego nie zaleca się maksymalnego obciążania poszczególnych urządzeń, a przeciwnie, podczas projektowania systemu należy pozostawić pewien zapas mocy obliczeniowej, aby dać możliwość szybkiej reakcji w przypadku maksymalnego obciążenia. Np. w systemie sterowania reaktorem jądrowym, w przypadku wystąpienia awarii reaktora zapotrzebowanie na moc obliczeniową może być znacznie większe niż podczas jego normalnej pracy. Jeśli system czasu rzeczywistego nie został zaprojektowany z uwzględnieniem maksymalnego obciążenia, to podczas awarii reaktora może mieć miejsce wadliwe funkcjonowanie systemu sterowania.

4.1.4. Zastosowanie systemów wieloprocesorowych

W wieloprocesorowych systemach komputerowych wiele zadań może być wykonywanych jednocześnie (równolegle). W chwili obecnej także

w mikrokomputerach są stosowane różne architektury systemów wieloprocesorowych. Jako przykład można podać technologie HT (ang. Hyper Treading) oraz Multicore firmy Intel.

Wieloprocesorowość systemu znacznie komplikuje algorytmy zarządzania zasobami komputera. Złożoność wynika z większej liczby konfliktów występujących podczas dostępu do urządzeń WE/WY, danych, pamięci, jak również konieczności blokowania zadań podczas dostępu do udostępnianych struktur jądra. Wszystkie te problemy muszą być rozwiązane przez system operacyjny poprzez synchronizację procesów i wątków, organizację kolejek oraz planowanie zasobów. Ponadto sam system operacyjny musi być tak zaprojektowany, aby ograniczyć wzajemne zależności między własnymi komponentami.

4.2. Zarządzanie procesami i wątkami

Podsystem zarządzania procesami i wątkami jest zaliczany do głównych podsystemów wielozadaniowego systemu operacyjnego i w znacznym stopniu wpływa na funkcjonowanie systemu komputerowego. Do głównych zadań tego podsystemu należy zaliczyć: tworzenie oraz usuwanie procesów i wątków, obsługa ich współdziałania, a także udostępnianie czasu procesora jednocześnie wykonywanym procesom i wątkom.

Podsystem zarządzania procesami i wątkami jest odpowiedzialny za przydzielanie procesom niezbędnych zasobów. System operacyjny przechowuje w pamięci specjalne struktury danych, w których zapisuje informacje o zasobach przydzielonych każdemu procesowi. Dany zasób może być przydzielony do wyłącznego użytkowania przez proces lub do współdzielonego użytkowania przez wiele procesów. Niektóre zasoby są przydzielane procesowi w momencie jego tworzenia, a inne dynamicznie, podczas realizacji programu. Ponadto zasoby mogą być przypisane procesowi na cały czas życia procesu lub tylko na określony okres. Podczas przydzielania poszczególnych zasobów procesowi podsystem zarządzania procesami i wątkami współpracuje z innymi podsystemami systemu operacyjnego, takimi jak podsystem zarządzania pamięcią, zarządzania WE/WY, system plików.

Gdy w systemie operacyjnym wykonywanych jest wiele zadań, to mogą one być w pełni niezależne i wykonywać się asynchronicznie (zadania niezależne nie współdzielą zasobów z innymi zadaniami). Może także wystąpić sytuacja, gdy poszczególne zadania wykorzystują wspólne dane (zadania zależne). W takim przypadku istnieje konieczność synchronizowania zadań. Synchronizację wątków zalicza się do głównych zadań podsystemu zarządzania procesami i wątkami.

Po zakończeniu procesu system operacyjny wykonuje operacje związane ze zwolnieniem zasobów używanych przez ten proces. Podsystem zarządzania procesami i wątkami zamyka wszystkie pliki otwarte, zwalnia obszary pamięci zajmowane przez proces oraz systemowe struktury informacyjne o procesie.

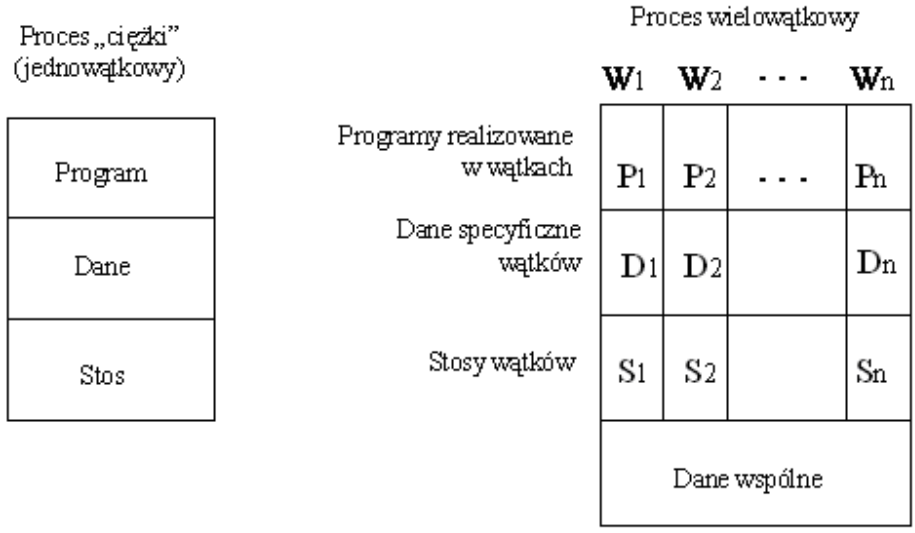
Ponadto korygowane są wszystkie kolejki oraz listy zasobów w systemie operacyjnym, w których występowały odwołania do zakończonego procesu.

4.2.1. Pojęcia „proces" i „wątek"

Aby wielozadaniowy system operacyjny mógł współbieżnie wykonywać wiele zadań, należy w systemie określić wewnętrzne jednostki wykonawcze, którym będzie przydzielany procesor. We współczesnych wielozadaniowych systemach operacyjnych stosowane są dwa typy jednostek wykonawczych – *procesy* oraz *wątki*. Do połowy lat 80. systemy operacyjne wykorzystywały wyłącznie pojęcie procesu. Współcześnie klasyczny proces określa się mianem procesu jednowątkowego lub procesu „ciężkiego". Proces jednowątkowy to wykonujący się program wraz z następującymi zasobami, przydzielonymi procesowi przez system operacyjny:

- pamięć operacyjna, w której przechowywany jest kod programu,
- przetwarzane dane oraz stos,
- pliki otwarte w procesie,
- inne zasoby informacyjne, jak gniazda TCP/UDP, muteksy, semafory itp.,
- dostęp do urządzeń WE/WY niezbędnych do realizacji programu,
- a przede wszystkim czas procesora niezbędny do realizacji programu.

W ramach wirtualnej przestrzeni adresowej przydzielonej procesowi przez system operacyjny można wyróżnić trzy części logiczne, określane mianem segmentów: segment programu (kodu), danych oraz stosu (rys. 4.3).



Rys. 4.3. Proces jednowątkowy („ciężki") oraz proces wielowątkowy

W wielozadaniowym systemie operacyjnym, w pamięci operacyjnej przechowywanych jest wiele procesów, z których każdy posiada własną wirtualną przestrzeń adresową. Poszczególne procesy są od siebie w pełni odizolowane, dzięki czemu program wykonywany w jednym procesie nie może odczytywać ani modyfikować danych innego procesu. Izolacja procesów jest jednym z głównych zadań systemu operacyjnego. Na skutek tego system funkcjonujący w oparciu o procesy ciężkie, charakteryzuje się wysoką stabilnością ponieważ nieprawidłowo wykonujący się jeden z procesów nie może zakłócić wykonywania procesów pozostałych.

Procesy niewykorzystujące wspólnych danych, a także niekomunikujące się ze sobą, określamy mianem *niezależnych.* Procesy niezależne mogą być przełączane przez system operacyjny w dowolnej kolejności, a wyniki ich działania są *deterministyczne*. Pod tym pojęciem rozumiemy fakt, że każdorazowe wykonanie programu dla określonych danych da zawsze ten sam wynik końcowy, mimo że podczas różnych wykonań system operacyjny inaczej przełącza procesor między procesami. Dzięki tej właściwości programista pisząc program, który będzie wykonywany w wielozadaniowym systemie operacyjnym, postępuje tak, jakby program ten miał być wykonywany w jednozadaniowym.

W systemie komputerowym bardzo często występuje sytuacja, gdy procesy wykorzystują wspólne dane, a także komunikują się ze sobą (np. procesy korzystające ze wspólnego pliku, procesy klientów bazy danych przetwarzające jednocześnie wspólne dane itp.). Procesy takie określa się mianem *zależnych*. Wyniki działania procesów zależnych w wielozadaniowym systemie operacyjnym są *niedeterministyczne*, gdyż zależą od sekwencji przełączania procesów przez system operacyjny (z punktu widzenia przetwarzania danych może być istotne czy w pierwszej kolejności procesor zostanie przydzielony procesowi P1, a później procesowi P2, czy odwrotnie). W takich przypadkach niezbędne jest użycie jednego z mechanizmów synchronizacji procesów, takich jak *muteksy*, *semafory* czy *sekcje krytyczne*. W systemach operacyjnych istnieje wiele mechanizmów komunikacji międzyprocesowej. Do podstawowych należy zaliczyć *potoki* (nienazwane oraz nazwane), *pamięć współdzieloną*, *wymianę komunikatów*, *gniazda*.

W systemach operacyjnych działających w oparciu o procesy jednowątkowe nie ma możliwości organizacji obliczeń równoległych w ramach jednego procesu. A taka potrzeba często istnieje ze względu na wygodę użytkownika czy też efektywne wykorzystanie zasobów systemu komputerowego. Procesy wielowątkowe, wprowadzone w drugiej połowie lat 80., dają możliwość obliczeń równoległych wykonywanych w ramach jednego procesu. Każdy z wątków wykonuje (współbieżnie z innymi wątkami tego procesu oraz pozostałymi wątkami w systemie operacyjnym) swój program, posiada swój stos oraz wykorzystuje własne dane (dane specyficzne wątku), a także dane globalne procesu (rys. 4.3).

Wszystkie wątki uruchomione w danym procesie mogą wykonywać ten sam program lub odwrotnie – każdy wątek może wykonywać inny program, zapisany w postaci odrębnego podprogramu. Wątki danego procesu wykonują się we wspólnej przestrzeni adresowej. Powoduje to, że poszczególne wątki mogą korzystać ze wspólnych danych (dane globalne procesu) co oznacza, że poszczególne wątki tego samego procesu są zależne. Z tego powodu mechanizmy synchronizacji wątków mają bardzo istotne znaczenie w wielowątkowych systemach operacyjnych.

Jako przykład aplikacji wielowątkowej, zwiększającej wygodę pracy użytkownika, można podać edytor tekstu, w którym jednocześnie możemy wykonywać następujące operacje: wyszukiwanie określonej frazy w tekście, wydruk dokumentu, zapis tekstu na dysku itp. W tym przypadku do każdej z powyższych operacji powoływany jest odrębny wątek wykonujący inny podprogram.

Aplikacje wielowątkowe są bardzo często stosowane w przypadku serwerów udostępniających określone usługi. W takim przypadku po zgłoszeniu się klienta do serwera tworzony jest nowy wątek, wykonujący program serwera (oznacza to, że wszystkie wątki realizują ten sam program). Takie rozwiązanie jest bardzo efektywne ze względu na zasoby systemu komputerowego, a szczególnie pamięci, ponieważ wszystkie wątki wykonują program, który jest w pamięci procesu zapisany w jednym egzemplarzu. Ponadto serwer wielowątkowy pracuje szybciej niż serwer pracujący wieloprocesowo ze względu na fakt, że operacja przełączania wątków odbywa się znacznie szybciej niż przełączanie procesów.

4.2.2. Tworzenie procesów i wątków

Podczas tworzenia procesu system operacyjny zapisuje wszystkie informacje o nowym procesie, takie jak identyfikator procesu, dane o rozmieszczeniu w pamięci, poziom uprzywilejowania itp., w odpowiednich strukturach danych (w systemie UNIX – deskryptor procesu, a w Windows NT – obiekt-proces). Powstanie nowego deskryptora procesu w systemie UNIX oznacza, że pojawił się nowy pretendent do zasobów systemu komputerowego. Od tego momentu system operacyjny podczas udostępniania zasobów komputera musi uwzględnić potrzeby nowego procesu.

W wyniku stworzenia nowego procesu system operacyjny przydziela procesowi określony obszar pamięci, a następnie ładuje w to miejsce zawartość odpowiedniego pliku wykonywalnego z pamięci masowej (obraz procesu) oraz, jeśli istnieje taka konieczność, dokonuje odpowiednich zmian w programie, zależnie od rozmieszczenia w pamięci. W tym zakresie podsystem zarządzania procesami ściśle współpracuje z podsystemami zarządzania pamięcią oraz systemem plików.

W systemie wielowątkowym, po stworzeniu procesu, system operacyjny tworzy co najmniej jeden wątek wykonania. Podczas tworzenia wątku,

analogicznie jak podczas tworzenia procesu, system operacyjny generuje specjalną strukturę informacyjną zawierającą między innymi identyfikator wątku, informację o prawach dostępu i przydzielonym priorytecie oraz o aktualnym stanie wątku. W pozycji wyjściowej wątek (lub proces w systemie operacyjnym obsługującym wyłącznie procesy jednowątkowe) znajduje się w stanie wstrzymania. Moment wyboru wątku do wykonania odbywa się zgodnie z przyjętym w systemie operacyjnym algorytmem szeregowania wątków, na podstawie aktualnego stanu wszystkich wątków w systemie.

Dany wątek może zwrócić się do systemu operacyjnego o stworzenie nowego wątku potomnego. W systemach operacyjnych stosowane są różne zasady współdziałania wątków macierzystych i wątków potomnych. Np. w jednych systemach operacyjnych wykonywanie wątku macierzystego jest zsynchronizowane z jego potomkami, a w szczególności po zakończeniu wątku macierzystego kończone są wszystkie wątki potomne. W innych systemach operacyjnych wątki potomne mogą wykonywać się asynchronicznie w stosunku do wątku rodzicielskiego.

Jako przykład rozpatrzmy operację tworzenia procesu w powszechnie stosowanej wersji V Release 4 systemu UNIX. W tej wersji systemu nie były stosowane wątki, a jednostką przydziału procesora był proces. Do zarządzania procesami system operacyjny wykorzystuje trzy podstawowe struktury danych związane z procesem: deskryptor procesu, u-obszar oraz tzw. sprzętowy kontekst procesu. Struktury te określane są mianem *kontekstu procesu*. *Deskryptor procesu* zawiera informacje o procesie niezbędne dla jądra, które są dostępne podczas całego cyklu życia procesu, niezależnie od stanu, w jakim aktualnie proces się znajduje, a także od tego czy jest aktualnie w pamięci, czy też został przeniesiony na dysk.

Do głównych pól deskryptora procesu można zaliczyć:

- stan procesu,
- informacje pozwalające na zlokalizowanie procesu i jego u-obszaru w pamięci głównej lub masowej,
- identyfikator procesu PID (ang. Process Identifier),
- identyfikatory procesów spokrewnionych,
- identyfikator użytkownika UID (ang. User Identifier), który utworzył proces,
- deskryptor zdarzenia, gdy proces jest w stanie uśpionym,
- parametry szeregowania,
- pole sygnałów (sygnały wysłane do procesu i jeszcze nieodebrane),
- różnego rodzaju liczniki wykorzystywane do celów rozliczeniowych oraz wyliczania priorytetu procesu.

Deskryptory poszczególnych procesów są przechowywane w przestrzeni pamięci przeznaczonej dla jądra, w postaci tablicy procesów. W oparciu o informacje zawarte w tablicy procesów system operacyjny wykonuje operacje związane z szeregowaniem i synchronizowaniem procesów. Część informacji o procesie system operacyjny UNIX przechowuje w u-obszarze, zapisanym

z wirtualnej przestrzeni adresowej systemu operacyjnego, lecz przechowywanym wraz z obrazem procesu. Do obszaru tego system ma dostęp wyłącznie w momencie, gdy dany proces jest aktywny.

Ze względu na to, że proces może być wykonywany w trybie użytkownika (realizacja programu użytkowego) oraz w trybie jądra (realizacja wywołań systemowych), w procesie jest przechowywany stos dla trybu użytkownika oraz dla trybu jądra. Poprzez te stosy są przekazywane parametry od programu użytkowego do funkcji systemowych wykonywanych w trybie jądra. Trzeba przy tym zwrócić uwagę, że wykonywanie funkcji systemowej odbywa się w kontekście danego procesu (podczas wywołania funkcji systemowej nie ma miejsca przełączenie kontekstu).

Sprzętowy kontekst procesu zawiera zbiór informacji ściśle związanych z konkretną platformą sprzętową, a w szczególności z architekturą procesora, niezbędnych do kontynuowania procesu, poczynając od miejsca, w którym wcześniej system operacyjny przerwał jego wykonywanie. W skład sprzętowego kontekstu procesu wchodzą następujące informacje:

- stan licznika rozkazów,
- stan rejestru flagowego,
- stany rejestrów adresowych danych oraz stosu,
- zawartości rejestrów roboczych procesora.

Sprzętowy kontekst procesu, analogicznie do deskryptora procesu oraz u-obszaru, dostępny jest wyłącznie dla programów jądra, czyli jest zapisany w wirtualnej przestrzeni adresowej systemu operacyjnego, jednak jest przechowywany wraz z obrazem procesu.

Tworzenie nowego procesu w systemie UNIX odbywa się poprzez wywołanie funkcji systemowej `fork()`. Proces wywołujący tę funkcję określany jest mianem procesu macierzystego, natomiast nowo utworzony – procesem potomnym. Funkcja tworzy nowy proces będący „klonem" procesu macierzystego. Proces potomny posiada identyczny kontekst jak proces macierzysty, lecz ma przydzielony inny identyfikator. Od tego momentu w obydwu procesach wykonywane są współbieżnie te same programy, poczynając od następnej instrukcji po funkcji `fork()`. Funkcja `fork()` wraca różne wartości w procesie macierzystym i procesie potomnym: w procesie macierzystym – identyfikator procesu potomnego, w procesie potomnym – 0 (zero). Uproszczony algorytm realizowany przez funkcję `fork()` można przedstawić następująco:

```
wejście: brak
wyjście:      w procesie macierzystym PID (ang.
              Process Identifier) potomka
              w procesie potomnym 0
{
sprawdź dostępność zasobów jądra;
pobierz wolną pozycję w tablicy procesów, określ
   unikatowy PID;
```

```
sprawdź, czy użytkownik nie wykonuje zbyt wielu
   procesów;
zaznacz, że potomek jest w stanie „tworzony”;
skopiuj dane z pozycji w tablicy procesów
   związanej z procesem macierzystym do pozycji
   związanej z procesem potomnym;
utwórz w pamięci kopię kontekstu procesu
   macierzystego (u-obszar, program, dane, stos);
if (wykonywany proces jest procesem macierzystym)
{
   zmień stan potomka na „gotowy do wykonania”;
   return(PID potomka);
}
else
{
   return(0);
}
}
```

W wyniku wykonania funkcji `fork()`, w procesie potomnym wykonywany jest ten sam program co w procesie macierzystym, co z praktycznego punktu widzenia wydaje się bezsensowne. Stworzenie nowego procesu ma praktyczne znaczenie jedynie w przypadku, gdy w procesie potomnym zostanie wykonany nowy program. W systemie UNIX jest to realizowane przy pomocy wywołania jednej z funkcji należących do grupy `exec()`, które powodują załadowanie do procesu nowego programu, zapisanego na dysku w postaci pliku wykonywalnego. W związku z tym programy wywołujące funkcję systemową `fork()` posiadają najczęściej następującą budowę:

```
pid=fork();
if (pid == 0)
   {operacje wykonywane w procesie potomnym
   exec(plik zawierający obraz nowego procesu)}
   else
   {operacje wykonywane w procesie macierzystym}
```

Współbieżne wykonywanie procesu macierzystego i potomnego może być realizowane na dwa sposoby:

- procesy wykonują się w pełni niezależnie,
- proces macierzysty, po powołaniu do życia procesu potomnego, wstrzymuje wykonanie i oczekuje na zakończenie pracy potomka.

W drugim przypadku w procesie macierzystym, po wykonaniu funkcji `fork()`, wykonana zostaje funkcja `wait()` powodująca wstrzymanie biegu procesu i oczekiwanie na zakończenie pracy procesu potomnego.

Proces tworzenia wątku jest znacznie mniej pracochłonny od tworzenia nowego procesu. Proces tworzenia nowego wątku prześledzimy na przykładzie

systemu operacyjnego Windows NT. Do tworzenia nowego wątku w systemie operacyjnym przewidziano funkcję systemową `CreateThread()`. Nieco upraszczając, można powiedzieć, że funkcja ta realizuje następujące operacje:

- tworzenie stosu trybu użytkownika dla nowego wątku w przestrzeni adresowej procesu,
- inicjalizacja sprzętowego kontekstu wątku,
- tworzenie i inicjalizacja bloku wykonawczego wątku (ang. ETHREAD – Executive Thread Block), zawierającego podstawowe informacje o wątku,
- inkrementacja licznika wątków w bloku wykonawczym procesu (ang. EPROCESS – Executive Process Block),
- określenie identyfikatora nowego wątku,
- inicjalizacja stosu trybu jądra dla nowego wątku,
- wyznaczenie uchwytu (ang. Handle) do nowo utworzonego wątku i zwrócenie tego uchwytu do programu wykonującego wywołanie funkcji CreateThread().

4.2.3. Planowanie i przełączanie wątków

W całym cyklu życia procesu wykonywanie wątków wchodzących w skład tego procesu może być wielokrotnie przerywane. (W systemie operacyjnym, pracującym wyłącznie w oparciu o procesy jednowątkowe, wszystkie poniższe uwagi dotyczą całego procesu). Przejście od wykonywania jednego wątku do drugiego w systemie operacyjnym jest realizowane dzięki planowaniu oraz przełączaniu wątków. Operacja planowania określa:

- moment, w którym zostaje przerwane wykonywanie bieżącego wątku,
- który z wątków będzie mógł być wykonywany w następnej kolejności.

Planowanie wątków odbywa się na podstawie informacji zapisanych w odpowiednich strukturach danych charakteryzujących poszczególne wątki i procesy, a w szczególności: priorytetu procesu i wątku, czasu oczekiwania w kolejce na przydzielenie procesora, sumarycznego okresu czasu, w którym wykonywany jest dany wątek, intensywności operacji WE/WY. System operacyjny planuje wykonywanie wątków niezależnie od tego czy wątki należą do jednego procesu, czy też różnych. W ten sposób po zakończeniu wykonywania wątku jednego procesu może nastąpić przełączenie do innego wątku tego samego procesu lub wątku należącego do innego procesu.

Istnieje cały szereg algorytmów planowania wątków, realizujących różne cele i charakteryzujących się różnymi właściwościami. W większości współczesnych systemów operacyjnych operacja planowania wątków jest realizowana dynamicznie (ang. on-line). Oznacza to, że decyzje dotyczące przełączania wątków podejmowane są podczas pracy systemu, na podstawie aktualnych informacji o procesach i wątkach. W takiej sytuacji system operacyjny pracuje w warunkach nieokreśloności, gdyż procesy i wątki są tworzone oraz kończone w sposób przypadkowy. W celu określenia optymalnej

kolejności realizacji wątków (zgodnie z założonym kryterium), system operacyjny wymaga określonego czasu procesora.

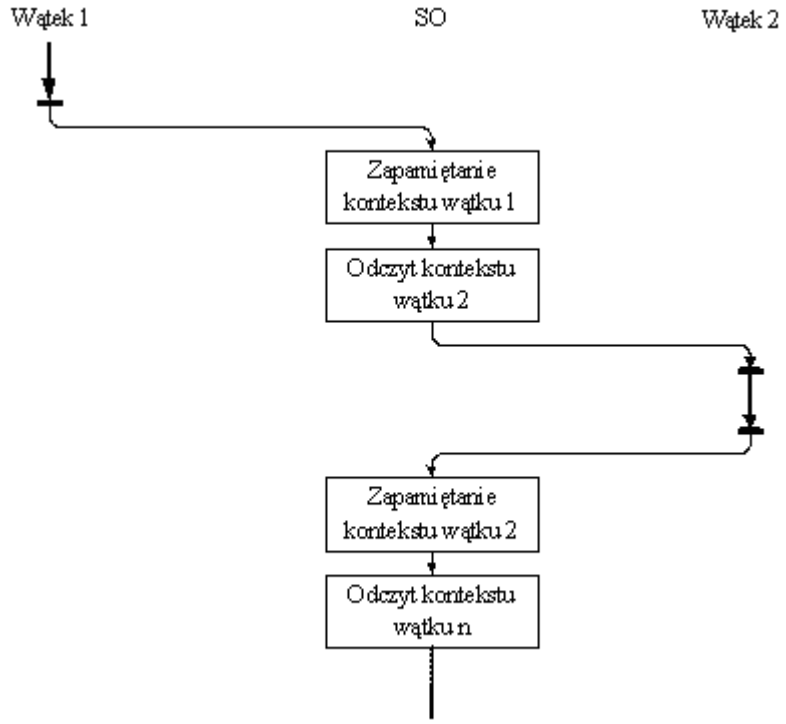
Statyczne planowanie wątków może być stosowane w specjalistycznych systemach operacyjnych, w których liczba wykonywanych procesów i wątków jest określona już na etapie projektowania. W takim przypadku algorytm przełączania wątków może być określony off-line, podczas tworzenia systemu operacyjnego.

Operacja przełączania wątków jest związana z przełączeniem procesora z jednego wątku na drugi. Przed zakończeniem realizacji jednego wątku system operacyjny zapamiętuje jego kontekst, aby móc w przyszłości realizować ten wątek od momentu, w którym został przerwany.

Przełączanie wątków sprowadza się do następujących operacji (rys. 4.4):

- zapamiętanie kontekstu bieżącego wątku,
- odczyt kontekstu nowego wątku wybranego przez program szeregujący do wykonania,
- rozpoczęcie realizacji nowego wątku.

Ponieważ przełączenie kontekstu wymaga wielu operacji, producenci procesorów wyposażają je w mechanizmy sprzętowego wspomagania przełączania kontekstu.



Rys. 4.4. Przełączanie kontekstu

4.2.4. Stany procesów i wątków

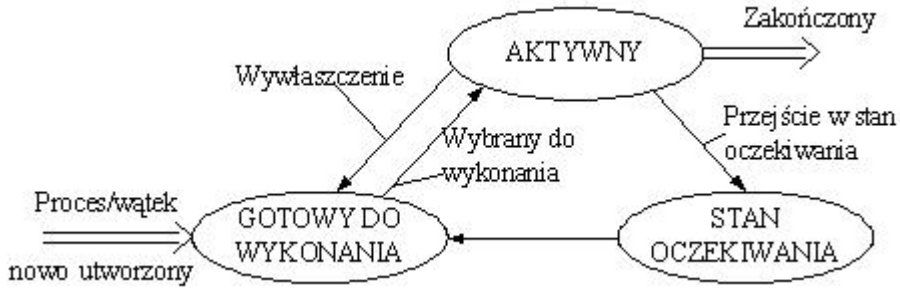
Podczas planowania procesów i wątków system operacyjny bierze pod uwagę ich stany. Ogólnie biorąc proces lub wątek może być w jednym z trzech poniższych stanów:

- aktywny - aktualnie wykonywany przez procesor,
- gotowy do wykonania – pasywny, oczekujący na przydzielenie procesora,
- oczekujący – pasywny, zablokowany ze względu na przyczyny wewnętrzne (np. trwa operacja WE/WY, oczekiwanie na odpowiedź od innego procesu/wątku, oczekiwanie na zwolnienie określonego zasobu komputera).

Na rys. 4.5 przedstawiono graf stanów procesu/wątku oraz możliwe przejścia między stanami. Nowo utworzony proces/wątek znajduje się w stanie „gotowy do wykonania", co oznacza, że został postawiony w kolejce procesów/wątków gotowych do wykonania. Po podjęciu decyzji przez planistę o przydzieleniu procesora, dany proces/wątek przechodzi w stan aktywny. W stanie tym pozostaje do momentu, aż:

- sam przejdzie w stan oczekiwania na określone zdarzenie,
- zostanie wywłaszczony przez program szeregujący i przejdzie w stan „gotowy do wykonania".

Po wystąpieniu określonego zdarzenia (np. zakończenie operacji WE/WY, zwolnienie określonego zasobu itp.) proces/wątek przechodzi ze stanu „oczekiwanie" do stanu „gotowy do wykonania".



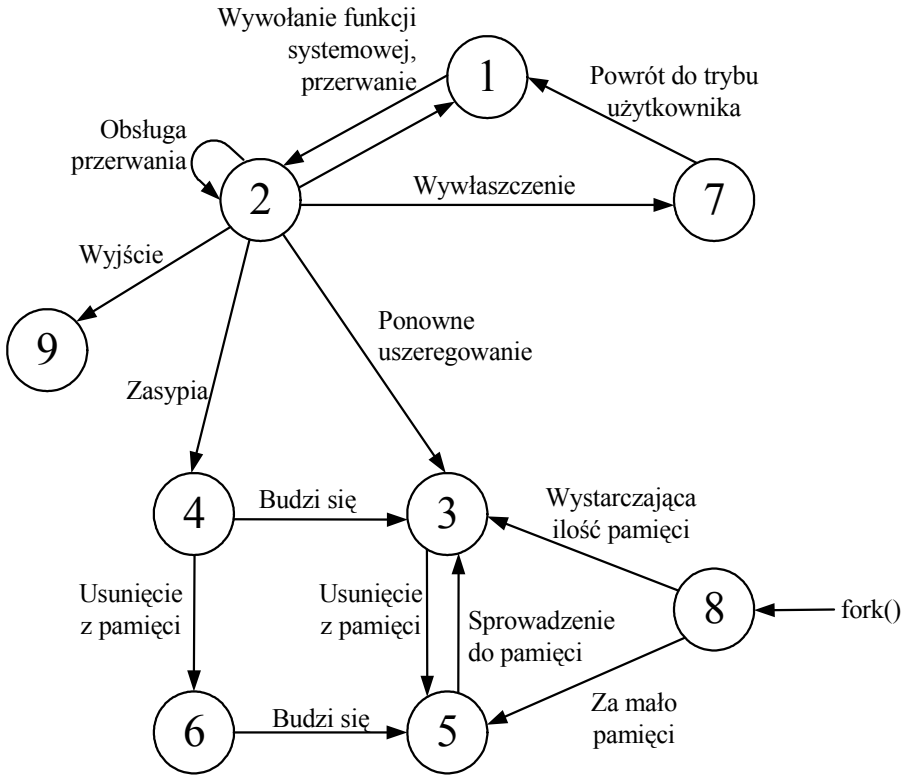
Rys. 4.5. Stany procesu/wątku oraz możliwe zmiany stanów

Jednoczesna liczba procesów/wątków znajdująca się w stanie aktywnym może być co najwyżej równa liczbie procesorów znajdujących się w danym systemie komputerowym. W rzeczywistości liczba możliwych stanów procesów/wątków jest znacznie większa niż przedstawiona na rys. 4.5. Dla przykładu rozpatrzmy stany procesów oraz diagram przejść między stanami w systemie UNIX wersji V oraz w systemie Windows NT.

W systemie UNIX wersji V występują następujące stany procesów:

- 1. wykonywany w trybie użytkownika,
- 2. wykonywany w trybie jądra,
- 3. gotowy do wykonania w pamięci,
- 4. uśpiony w pamięci,
- 5. gotowy do wykonania poza pamięcią,
- 6. uśpiony poza pamięcią,
- 7. wywłaszczony,
- 8. utworzony,
- 9. zombie.

Na rys. 4.6 przedstawiono pełen diagram przejść procesu między poszczególnymi stanami.



Rys. 4.6. Stany procesów w systemie UNIX

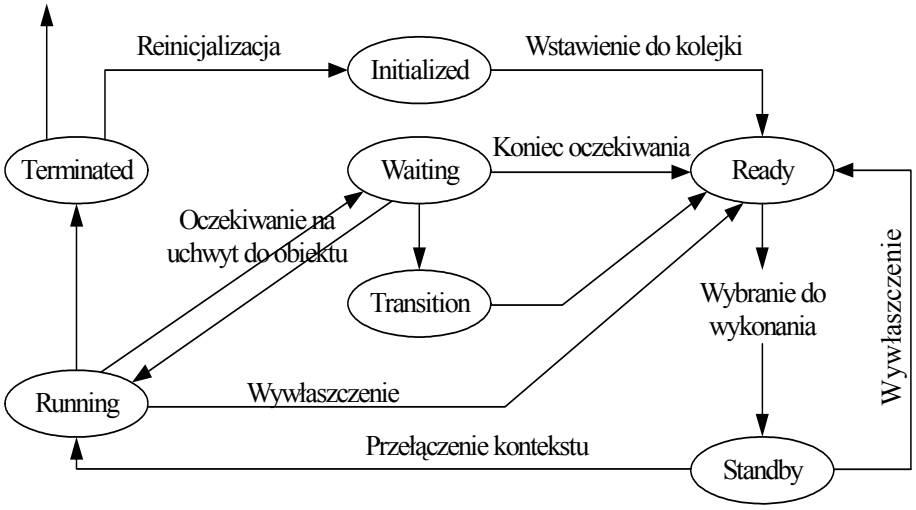
Proces pojawia się w systemie w stanie „utworzony” po wykonaniu przez proces macierzysty funkcji `fork()`, a następnie – zależnie od aktualnej zajętości pamięci operacyjnej - przechodzi do stanu „gotowy do wykonania w pamięci” lub „gotowy do wykonania poza pamięcią” (stany 3 oraz 5). Stan procesu „wykonywany w trybie jądra” jest związany z przełączaniem kontekstu procesu oraz wywoływaniem przez proces funkcji systemowych (operacje te

wykonywane są w trybie uprzywilejowanym jądra systemu operacyjnego). Stany „wywłaszczony" oraz „gotowy do wykonania w pamięci" są w istocie tożsame i są związane z oczekiwaniem procesu na przydział procesora i przejściem w stan „wykonywany w trybie jądra". Proces „uśpiony" to proces w stanie oczekiwania na określone zdarzenie. W przypadku braku pamięci operacyjnej proces „uśpiony" może zostać usunięty z pamięci operacyjnej i w ten sposób przejść w stany „uśpiony poza pamięcią" oraz „gotowy do wykonania poza pamięcią". Szczególnym stanem procesu w systemie UNIX jest stan „zombie". Proces przechodzi do tego stanu w sytuacji, gdy wykonał funkcję systemową `exit()` i przestał istnieć, jednak pozostały po nim pewne informacje, jak np. kod wyjścia czy statystyki.

W systemie Windows NT występują następujące stany wątków:

- Ready – gotowy do wykonania,
- Standby – wybrany przez program szeregujący do wykonania,
- Running – wykonywany,
- Waiting – oczekujący na określone zdarzenie, np. zakończenie operacji WE/WY,
- Transition – gotowy do wykonania, lecz stos jądra tego wątku znajduje się poza pamięcią w wyniku operacji stronicowania,
- Terminated – zakończony.

Na rys. 4.7 przedstawiono stany wątków oraz możliwe przejścia między stanami.



Rys. 4.7. Stany wątków w SO Windows NT oraz przejścia między nimi

Podczas poszukiwania wątku do wykonania planista bierze pod uwagę jedynie wątki będące w stanie *Ready*. W stanie *Standby* znajduje się w danej chwili wyłącznie tyle wątków, ile procesorów jest zainstalowanych w danym komputerze. Wątek w stanie *Standby* jest przygotowany do przełączenia

kontekstu. Wątek pozostaje w stanie *Running* aż do wywłaszczenia przez wątek o wyższym priorytecie, wyczerpania limitu czasu, przejścia w stan oczekiwania na uchwyt do obiektu lub zakończenia pracy. Przejście wątku w stan *Wait* wynika z konieczności synchronizacji jego wykonywania z realizacją innych wątków oraz z pracą układów WE/WY. Po zakończeniu oczekiwania wątek może kontynuować swoje wykonywanie lub przejść w stan gotowości do wykonania (*Ready*). Wątek może przejść w stan przejściowy (*Transition*), gdy jest gotowy do wykonania, jednak jego stos trybu jądra został usunięty z pamięci w wyniku stronicowania pamięci. Po sprowadzeniu do pamięci brakujących stron wątek przechodzi w stan gotowości (*Ready*).

4.2.5. Algorytmy planowania

Algorytmy planowania wątków można podzielić na dwie grupy:

- *bez wywłaszczania* (ang. non-preemptive), polegające na tym, że aktywny wątek wykonuje się tak długo, aż sam zrezygnuje z procesora i przekaże sterowanie do systemu operacyjnego, który wybierze inny wątek do wykonania,
- *z wywłaszczaniem* (ang. preemptive), w których decyzja o przełączeniu wątku podejmowana jest przez system operacyjny, a nie przez aktywny wątek.

Podstawowa różnica między obydwoma algorytmami polega na stopniu centralizacji mechanizmu szeregowania wątków. W przypadku algorytmów z wywłaszczeniem wszystkie funkcje planowania wątków są skupione w systemie operacyjnym, a programista piszący program użytkowy nie interesuje się problemem przełączania wątków. System operacyjny realizuje w takim przypadku następujące funkcje:

- określa moment zakończenia realizacji aktywnego wątku,
- zapamiętuje jego kontekst,
- wybiera z kolejki wątków gotowych do wykonania kolejny wątek,
- odczytuje kontekst nowego wątku,
- rozpoczyna realizację tego wątku.

W przypadku algorytmów bez wywłaszczania, mechanizm szeregowania wątków jest rozłożony między systemem operacyjnym i programami użytkowymi. Program użytkowy po przejęciu sterowania od systemu operacyjnego sam określa moment zakończenia realizacji wątku i przekazuje sterowanie do systemu operacyjnego za pośrednictwem odpowiedniego wywołania systemowego. System operacyjny tworzy kolejkę wątków gotowych do wykonania i wybiera, zgodnie z przyjętym algorytmem planowania, kolejny wątek do wykonania. Taki sposób planowania wątków stwarza pewne problemy tak dla użytkownika, jak i dla programisty aplikacji. Z punktu widzenia użytkownika wykonanie programu w znacznym stopniu angażującego procesor (np. złożone obliczenia numeryczne) pociąga za sobą brak możliwości jednoczesnego wykonania innego programu. Z tego powodu twórcy programów

użytkowych dla systemów operacyjnych pracujących zgodnie z algorytmem bez wywłaszczania powinni tak tworzyć programy, aby wykonywały operacje pewnymi „porcjami" i cyklicznie przekazywać sterowanie do systemu operacyjnego, dając w ten sposób możliwość jednoczesnej realizacji innych programów. Stawia to przed programistami aplikacji użytkowych dodatkowe zadania. W przypadku systemów operacyjnych stosujących algorytm z wywłaszczaniem, programista tworzy programy użytkowe tak, jakby programy te miały być wykonywane w jednozadaniowym systemie operacyjnym, bez zastanawiania się nad problemem szeregowania wątków.

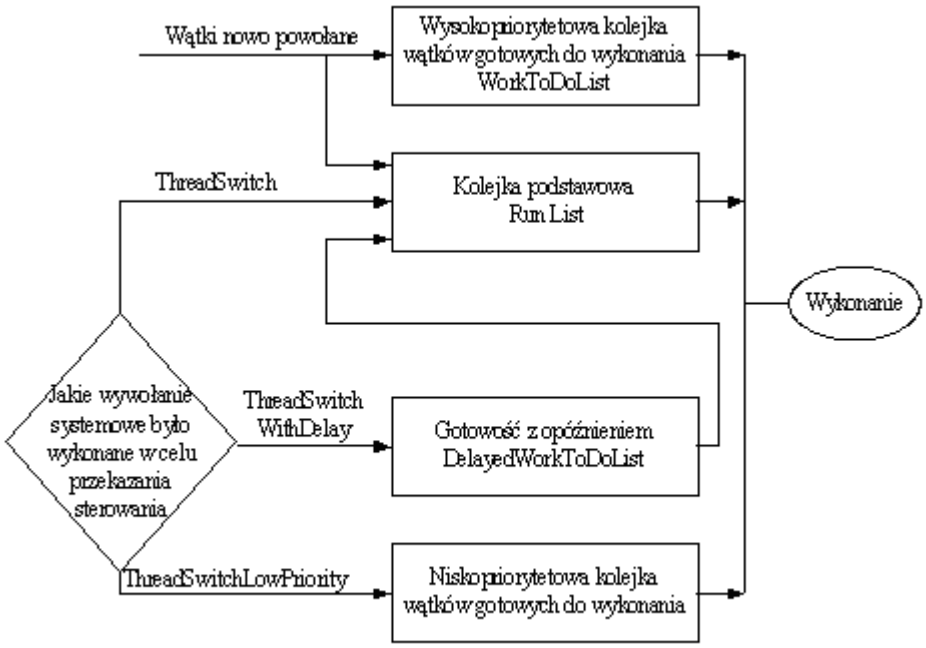
Mimo to w wielu systemach operacyjnych wykorzystywane są algorytmy planowania wątków bez wywłaszczania, gdyż pozwalają programiście aplikacji użytkowych dobierać sposób przełączania wątków do określonego zadania. Dodatkowo programista może w takim przypadku łatwo rozwiązywać problemy jednoczesnego wykorzystywania wspólnych danych przez wiele aplikacji, zapewniając wyłączność w dostępie do wspólnych danych przez określony wątek. Ponadto algorytm szeregowania wątków bez wywłaszczania charakteryzuje się większą szybkością i lepszym wykorzystaniem mocy obliczeniowej komputera.

Niemal wszystkie współczesne wielozadaniowe systemy operacyjne funkcjonują w oparciu o algorytmy z wywłaszczeniem. Przykładem efektywnego wykorzystania algorytmu szeregowania wątków bez wywłaszczania jest system operacyjny NetWare 4.x, w którym dzięki temu uzyskano znaczną szybkość wykonywania operacji plikowych. Wątek wykonywany w systemie NetWare sam przekazuje sterowanie do systemu operacyjnego, wykorzystując następujące funkcje systemowe:

- *ThreadSwitch* – wątek wywołujący tę funkcję informuje system operacyjny o tym, że może się wykonywać dalej, jednak przerywa swoje działanie aby dać możliwość wykonania innych wątków,
- *ThreadSwitchWithDelay* – funkcja analogiczna do poprzedniej, jednak wątek informuje system operacyjny, że będzie gotowy do wykonania po zadanej liczbie przełączeń wątków,
- *Delay* – funkcja analogiczna do poprzedniej, jednak czas gotowości wątku do wykonania podaje się w milisekundach,
- *ThreadSwitchLowPriority* – funkcja przekazująca sterowanie do systemu operacyjnego, różniąca się od funkcji ThreadSwitch tym, że wątek zostanie ustawiony do kolejki wątków o niskim priorytecie, gotowych do wykonania.

Planista systemu NetWare tworzy kilka kolejek wątków (rys. 4.8). Wątek nowo utworzony trafia na koniec kolejki *RunList*, zawierającej wątki gotowe do wykonania. Po przekazaniu sterowania do systemu operacyjnego wątek trafia na koniec odpowiedniej kolejki, w zależności od tego jaką funkcję systemową wykonano w wątku do przekazania sterowania. Do kolejki *RunList* trafiają wątki, w których wykonano funkcję *ThreadSwitch*, do kolejki *DelayedWorkToDoList* wątki, w których wykonano funkcję

ThreadSwitchWithDelay lub *Delay*, a do kolejki *LowPriorityRunList* wątki, w których wykonano funkcję *ThreadSwitchLowPriority*. Planista systemu NetWare wybiera do wykonania wątek znajdujący się na początku kolejki *RunList*, a gdy kolejka jest pusta - pierwszy wątek z kolejki *ThreadSwitchLowPriority*.



Rys. 4.8. Planowanie wątków w systemie NetWare

Po upływie czasu oczekiwania zadanego w wywołaniu funkcji *ThreadSwitchWithDelay* oraz *Delay* wątek trafia na koniec kolejki *RunList*. Kolejka *WorkToDoList* posiada najwyższy priorytet i trafiają do niej wątki związane z realizacją zadań systemowych.

4.2.6. Algorytmy planowania oparte na kwancie czasu (ang. quantum)

Algorytmy planowania wątków oparte na wywłaszczaniu wykorzystują pojęcie kwantu czasu, jako maksymalnego czasu, podczas którego wątek może wykonywać się nieprzerwanie. Wątek, który wykorzystał swój kwant czasu, zostaje przełączony do stanu gotowości i oczekuje aż zostanie mu przydzielony nowy kwant czasu. Uproszczony graf stanów wątku w takim przypadku przedstawiono na rys. 4.9.

Rys. 4.9. Stany wątków w systemach operacyjnych opartych na kwancie czasu

Kwanty czasu przydzielane poszczególnym wątkom mogą być jednakowe lub mogą mieć różne wartości. Ponadto kwant czasu może zmieniać się dla danego wątku podczas upływu czasu wykonywania wątku.

W przypadku gdy wszystkim wątkom przydzielony jest ten sam kwant czasu $\Delta\tau$, oraz w systemie występuje n wątków, to czas oczekiwania danego wątku na kolejny kwant czasu można w przybliżeniu określić następująco: $(n-1)*\Delta\tau$. Im więcej wątków występuje w systemie, tym dłuższy jest czas oczekiwania na procesor, a poszczególne wątki wykonują się wolniej. Zwiększanie kwantu czasu powoduje, że część wątków kończy swoje działanie podczas trwania pierwszego kwantu czasu, a zależność $(n-1)*\Delta\tau$ określająca czas oczekiwania wątku na przydział procesora przestaje być słuszna. W przypadku gdy kwant czasu jest dostatecznie długi, algorytm szeregowania oparty na wywłaszczaniu z kwantem czasu sprowadza się do kolejnego wykonania poszczególnych wątków.

Kwanty czasu przydzielane danemu wątkowi mogą także zmieniać się podczas cyklu życia wątku. W systemach operacyjnych często przyjmowana jest duża wartość początkowa kwantu czasu, a następnie każdy kolejny kwant jest zmniejszany. Daje to efekt szybszego kończenia wątków krótkich, co w rezultacie zwiększa przepustowość systemu operacyjnego, liczoną jako liczba wątków wykonanych w jednostce czasu.

Podczas wykonywania wątków nie wszystkie z nich wykorzystują w pełni przydzielony kwant czasu ze względu na wcześniejsze przejście w stan oczekiwania, np. ze względu na konieczność wykonania operacji WE/WY. Powstaje w związku z tym sytuacja, że wątki wykonujące intensywne operacje WE/WY wykorzystują tylko część przydzielonego czasu procesora. Jeśli dany wątek nie wykorzystał w pełni kwantu czasu, może zostać uprzywilejowany podczas późniejszego wykonywania. W tym celu planista systemu operacyjnego może stworzyć dwie kolejki wątków gotowych do wykonania (rys. 4.10). W kolejce nr 1 znajdują się wątki, które wyczerpały w pełni przydzielony kwant czasu, a w kolejce nr 2 – wątki, które przerwały działanie

przed upływem kwantu czasu. Podczas wyboru wątku do wykonania planista systemowy pobiera przede wszystkim wątki z kolejki nr 2.

Rys. 4.10. Planowanie wątków oparte na kwancie czasu z pierwszeństwem wątków wykonujących intensywne operacje WE/WY

4.2.7. Algorytmy planowania oparte na priorytetach

Inną ważną koncepcją leżącą u podstaw funkcjonowania współczesnych algorytmów planowania wątków z wywłaszczaniem jest obsługa priorytetowa. W takim przypadku każdemu wątkowi zostaje przydzielony priorytet, na podstawie którego planista systemu operacyjnego określa kolejność wykonywania wątków. Priorytet jest liczbą charakteryzującą stopień uprzywilejowania wątku podczas korzystania z zasobów systemu, a w szczególności z czasu procesora. Im wyższy priorytet posiada dany wątek, tym mniej czasu spędza w kolejkach do zasobów systemu.

Priorytet wątku jest z zasady bezpośrednio związany z priorytetem procesu, w ramach którego wykonuje się dany wątek. Priorytet procesu zostaje określony przez system operacyjny w momencie jego tworzenia, w zależności od wielu czynników: czy jest to proces systemowy, czy użytkowy, jaki status posiada użytkownik, który uruchomił ten proces itp. Priorytet wątku uruchomionego w ramach danego procesu jest określony przez system operacyjny na podstawie priorytetu procesu oraz parametrów wywołania systemowego tworzącego nowy wątek. W wielu systemach operacyjnych przewiduje się możliwość zmiany priorytetów podczas cyklu życia wątku. Priorytety takie określane są mianem dynamicznych.

W systemach operacyjnych są stosowane dwie odmiany priorytetowego planowania wątków: planowanie z priorytetami względnymi oraz bezwzględnymi. W obydwu przypadkach do wykonania wybierany jest wątek o najwyższym priorytecie, jednak odmiennie określany jest moment

zakończenia pracy wątku aktywnego. W systemach operacyjnych opartych na priorytetach względnych, wątek aktywny wykonuje się tak długo, aż sam zrezygnuje z procesora i przejdzie w stan oczekiwania (np. operacja WE/WY). Natomiast w systemach z priorytetami bezwzględnymi, pojawiający się nowy wątek o wyższym priorytecie od priorytetu wątku aktualnie wykonywanego powoduje wywłaszczenie wątku aktywnego i rozpoczęcie realizacji wątku o najwyższym priorytecie bezwzględnym.

4.2.8. Przykłady współczesnych algorytmów planowania wątków

W wielu współczesnych systemach operacyjnych algorytmy planowania wątków są oparte tak na koncepcji kwantu czasu, jak również priorytetach. Rozpatrzymy algorytmy planowania procesów/wątków w dwu systemach operacyjnych: UNIX System V Release 4 oraz Windows NT.

W systemie UNIX System V Release 4 nie występowały wątki, a planowanie odbywało się na poziomie procesów. Ogólnie można powiedzieć, że w tym systemie operacyjnym stosowany jest algorytm planowania z wywłaszczeniem, przy wykorzystaniu kwantu czasu oraz priorytetów. Każdy proces, w zależności od zadania jakie realizuje, jest przyporządkowany do jednej z trzech klas priorytetów: klasy czasu rzeczywistego, klasy procesów systemowych oraz klasy procesów z podziałem czasu. Przydzielenie oraz dalsze przetwarzanie priorytetów wykonywane jest w różny sposób dla poszczególnych klas. Dla procesów systemowych poziom priorytetu nadawany jest przez jądro systemu i jest niezmienny podczas całego cyklu życia procesu. Procesy czasu rzeczywistego także mają przydzielony stały poziom priorytetu, jednak może on być zmieniany przez użytkownika. Ponieważ procesom czasu rzeczywistego procesor przydzielany jest w pierwszej kolejności (procesy te posiadają najwyższy poziom priorytetu), więc należy je stosować ze szczególną ostrożnością, aby nie zablokować realizacji innych procesów. Planowanie procesów czasu rzeczywistego jest określone przez dwa parametry: poziom priorytetu globalnego oraz kwant czasu. Dla każdego poziomu priorytetu jest przypisany inny kwant czasu.

Klasa procesów z podziałem czasu jest klasą domyślną dla procesów nowo tworzonych. Dla tej klasy procesów planista systemowy stosuje algorytm priorytetów dynamicznych. Wielkość priorytetu dla danego procesu zostaje określona na podstawie dwóch składowych: części użytkownika oraz części systemowej. Część użytkownika w priorytecie może być zmieniona przez administratora systemu lub użytkownika, który jest właścicielem tego procesu. Jednak w drugim przypadku - wyłącznie w kierunku zmniejszenia priorytetu. Systemowa składowa priorytetu pozwala centralnemu planiście systemu operacyjnego na zarządzanie procesami w zależności od czasu, w jakim poszczególne procesy wykorzystywały procesor bez przejścia w stan oczekiwania. Procesom, które wykonują się dłużej bez przejścia w ten stan (co jest spowodowane np. operacją WE/WY), priorytet zostaje zmniejszany.

Procesom w mniejszym stopniu wykorzystującym kwant czasu (np. w wyniku intensywnych operacji WE/WY) priorytet jest podwyższany, na skutek czego częściej mają one przydzielony kolejny kwant czasu. Zmniejszenie priorytetu dla procesów, które w większym stopniu wykorzystują kwant czasu, jest kompensowane przez wydłużenie kwantu czasu, dzięki czemu procesy te mają rzadziej przydzielany procesor, za to na dłuższy okres.

W systemie Windows NT proces tworzy środowisko pracy wątków, udostępniając im swoje zasoby, a planista systemu operacyjnego przydziela procesor poszczególnym wątkom. Priorytet wątku w Win32 jest wypadkową dwóch wielkości: klasy priorytetowej procesu (ang. priority class) oraz priorytetu względnego wątku w ramach procesu. Klasa priorytetowa określa stopień preferencji procesu jako całości podczas ubiegania się jego poszczególnych wątków o czas procesora. W Win32 istnieją cztery klasy priorytetowe:

- jałowa, o wartości 4 (ang. Iddle) – proces opatrzony tą klasą otrzymuje czas procesora jedynie w przypadku, gdy inne procesy o wyższej klasie priorytetowej nie potrzebują procesora (np. wygaszacze ekranu),
- normalna, o wartości 8 (ang. Normal) – domyślna klasa priorytetowa przypisywana wszystkim procesom uruchamianym z pulpitu,
- wysoka, o wartości 13 (ang. High) – klasę tę otrzymują procesy, które powinny otrzymywać czas procesora tak szybko, jak to jest możliwe (np. lista zadań Windows, która niezależnie od innych czynności musi ukazać się użytkownikowi niezwłocznie),
- czasu rzeczywistego, o wartości 24 (ang. RealTime) – wątki procesu posiadającego tę klasę priorytetową zdolne są do wywłaszczania wątków wszystkich innych procesów, włącznie z procesami systemowymi.

Priorytet względny wątku pozwala zróżnicować poszczególne wątki w ubieganiu się o czas procesora. Danemu wątkowi można przypisać jeden z siedmiu następujących priorytetów względnych:

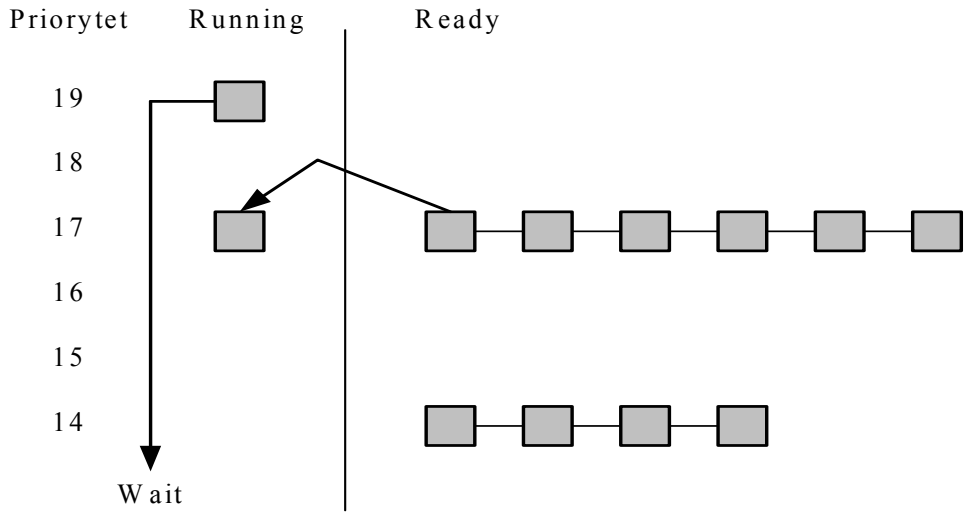
Priorytet	Waga liczbowa
- jałowy (tpIdle)	-15
- niski (tpLowest)	-2
- podnormalny (tpBelowNormal)	-1
- normalny (tpNormal)	0
- nadnormalny (tpAboveNormal)	+1
- wysoki (tpHighest)	+2
- krytyczny (tpTimeCritical)	+15

Wynikowy priorytet wątku jest równy sumie wartości klasy priorytetowej procesu oraz priorytetu względnego wątku. W szczególnych przypadkach wynikowy priorytet wątku przyjmuje inną wartość niż suma klasy priorytetowej procesu i priorytetu względnego wątku:

- wątek o priorytecie względnym tpIdle, należący do procesu o klasie priorytetowej innej niż RealTime, posiada ostateczny priorytet równy 1,
- wątek o priorytecie względnym tpIdle, należący do procesu o klasie priorytetowej RealTime, posiada ostateczny priorytet równy 16,
- wątek o priorytecie względnym tpTimeCritical, należący do procesu o klasie priorytetowej innej niż RealTime, posiada ostateczny priorytet równy 15,
- wątek o priorytecie względnym tpTimeCritical, należący do procesu o klasie priorytetowej RealTime, posiada ostateczny priorytet równy 31.

Centralny planista systemu Windows NT stosuje algorytm planowania z wywłaszczaniem, przy wykorzystaniu kwantu czasu oraz priorytetów. Algorytm planowania można scharakteryzować następująco:

Wątek może dobrowolnie zrzec się procesora poprzez wprowadzenie stanu oczekiwania na jakiś obiekt (np. na zdarzenie, muteks, semafor, zakończenie operacji WE/WY itd.) w drodze wywołania jednej z wielu funkcji oczekiwania Win32 (jak np. *WaitForSingleObject*). Rys. 4.11 ilustruje przejście wątku w stan oczekiwania i wybór przez system operacyjny nowego wątku do wykonania. Kwantum, które pozostało po przejściu wątku w stan oczekiwania, pozostaje bez zmian. Po zakończeniu stanu oczekiwania i ponownym przydziale procesora, kwant czasu zostaje zmniejszony o jednostkę.

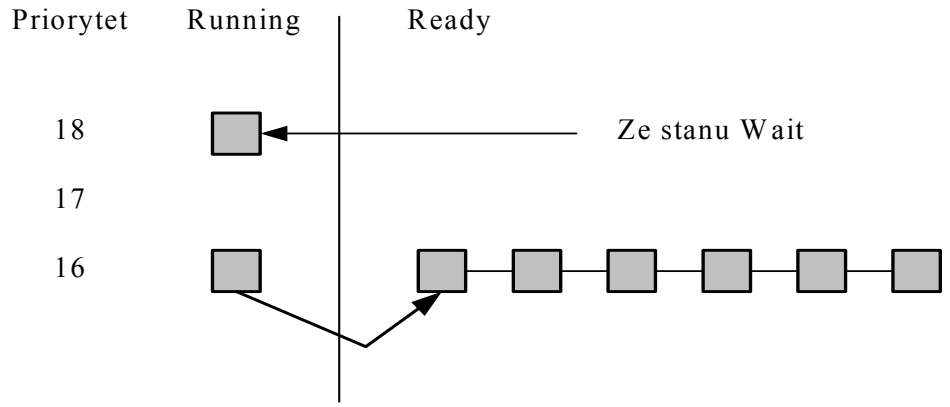


Rys. 4.11. Przykład planowania wątków w SO Windows NT

Aktywny wątek o niskim priorytecie zostaje wywłaszczony, gdy wśród wątków gotowych do wykonania pojawi się wątek o wyższym priorytecie. Przedstawiona sytuacja może wystąpić, gdy:

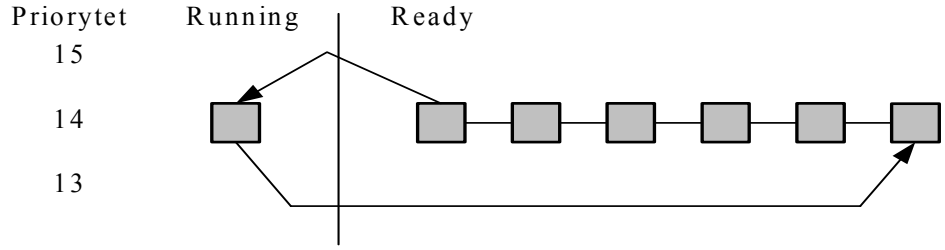
- wątek o wyższym priorytecie kończy oczekiwanie (przechodzi ze stanu *Wait* w stan *Ready*),
- zmieniony zostaje priorytet wątku (zwiększony w wątku oczekującym lub zmniejszony w wątku wykonywanym).

Po wywłaszczeniu wątek stawiany jest na początku kolejki wątków gotowych do wykonania dla priorytetu, z jakim był wykonywany, co zapewnia, że możliwie szybko będzie mógł wyczerpać swój limit czasu. Rys. 4.12 przedstawia sytuację, gdy wątek z priorytetem 18 przechodzi ze stanu oczekiwania w stan wykonania. Powoduje to wywłaszczenie wątku o priorytecie 16 i wstawienie go na początek kolejki wątków gotowych do wykonania z priorytetem 16.



Rys. 4.12. Przykład szeregowania wątków w SO Windows NT

Gdy wątek wyczerpuje kwant czasu system operacyjny musi zadecydować, czy priorytet wątku ma być zmniejszony, czy też nie. Jeśli priorytet wątku jest zmniejszany, to system wybiera wątek o wyższym priorytecie od priorytetu wątku aktualnie wykonywanego i przydziela mu procesor. W przypadku gdy priorytet procesu nie jest zmniejszany, system wybiera pierwszy wątek z kolejki wątków gotowych do wykonania o tym samym priorytecie, a wątek poprzednio wykonywany stawiany jest na końcu kolejki (z przydzielonym nowym kwantem czasu). Przypadek ten jest przedstawiony na rys. 4.13. Jeśli kolejka wątków o danym priorytecie, gotowych do wykonania, jest pusta, to dany wątek dostaje nowy kwant czasu i kontynuowane jest jego wykonywanie.



Rys. 4.13. Przykład szeregowania wątków w SO Windows NT

4.2.9. Planowanie wątków w systemach czasu rzeczywistego

W systemach czasu rzeczywistego, w których głównym kryterium efektywności jest spełnienie wymogów czasowych, planowanie procesów ma szczególne znaczenie. System czasu rzeczywistego powinien reagować na sygnały dostarczane od sterowanego obiektu z uwzględnieniem zadanych ograniczeń czasowych. Ponadto w systemie często podana jest informacja o czasach wykonania poszczególnych zadań, momentach aktywacji, zakresach dopuszczalnych czasów oczekiwania na odpowiedź itp. Dane te mogą być wykorzystywane przez centralnego planistę systemu do wyboru odpowiedniego algorytmu dynamicznego planowania wątków. Wymagania takie nie są stawiane klasycznym systemom operacyjnym. W zamian za to w systemach operacyjnych czasu rzeczywistego jest z zasady znana liczba procesów i wątków, co ułatwia planowanie.

Podczas opracowania algorytmów planowania wątków w systemie czasu rzeczywistego należy rozważyć jakie konsekwencje wynikają z niespełnienia narzuconych ograniczeń czasowych. Jeśli te konsekwencje mogą być katastroficzne, to system operacyjny, na podstawie którego jest budowany układ sterowania, jest określany mianem systemu „twardego” (ang. hard system). W przypadku gdy naruszenie ograniczeń czasowych nie pociąga za sobą katastroficznych konsekwencji, taki system czasu rzeczywistego określa się mianem „miękkiego” (ang. soft system).

W twardych systemach czasu rzeczywistego czas wykonania każdej krytycznej operacji musi być zagwarantowany dla wszystkich możliwych scenariuszy pracy systemu. Takich gwarancji można udzielić na podstawie pełnego przetestowania wszystkich możliwych scenariuszy realizowanych przez układ sterowania, opracowania statycznego harmonogramu realizacji poszczególnych programów lub w wyniku ścisłego matematycznego opracowania dynamicznego algorytmu szeregowania.

Podczas tworzenia statycznego harmonogramu realizacji poszczególnych wątków trzeba brać pod uwagę, że nie zawsze jest to możliwe dla zadanych ograniczeń czasowych. Podstawowym kryterium określającym realizowalność statycznego harmonogramu realizacji wątków jest warunek, aby różnica między krańcowym (ostatecznym) zadanym czasem wykonania zadania (po wystąpieniu zgłoszenia konieczności jego wykonania), a rzeczywistym czasem jego wykonania (przy założeniu ciągłego wykonywania tego zadania) była zawsze dodatnia. Jednak warunek ten jest tylko warunkiem koniecznym, a nie dostatecznym.

W miękkich systemach czasu rzeczywistego zakłada się, że może mieć miejsce przekroczenie zadanych ograniczeń czasowych podczas realizacji poszczególnych wątków lub procesów. Przykładami takich systemów mogą być: UNIX, Windows NT, OS/2 itd. Cechą charakterystyczną tych systemów operacyjnych jest możliwość tworzenia wątków/procesów o priorytecie czasu rzeczywistego (najwyższy priorytet w systemie operacyjnym). Gwarantuje to

szybkie wykonywanie takich wątków/procesów (pojawienie się wątku/procesu o najwyższym priorytecie powoduje wywłaszczenie wątku/zadania aktualnie wykonywanego), jednak nie gwarantuje spełnienia zadanych ograniczeń czasowych, charakterystycznych dla twardych systemów czasu rzeczywistego.

W zależności od charakteru występowania zgłoszeń wykonania zadań można wyróżnić dwa rodzaje zgłoszeń: periodyczne i sporadyczne. Dla zgłoszeń periodycznych (np. zegarowych) można ściśle określić ich momenty wystąpienia, natomiast dla zgłoszeń sporadycznych jest to niemożliwe. Załóżmy, że w systemie operacyjnym występuje zbiór zadań $\{T_i\}$ wywoływanych periodycznie z okresami p_i. Dla każdego z tych zadań jest określony dopuszczalny czas wykonania d_i, a czas realizacji każdego z zadań wynosi c_i. W celu sprawdzenia możliwości opracowania statycznego harmonogramu realizacji tych zadań wystarczy przeanalizować okres równy najmniejszej wspólnej wielokrotnej okresów p_i. Aby było możliwe opracowanie harmonogramu wykonania poszczególnych zadań, można podać następujący warunek (współczynnik $\Sigma c_i/p_i$ określany jest mianem sumarycznego współczynnika obciążenia procesora):

$\Sigma c_i/p_i <= k$, gdzie k - liczba procesorów w systemie.

Podczas wyboru algorytmu planowania wątków/procesów należy brać pod uwagę informacje o ewentualnej zależności zadań (zadania zależne to wątki/procesy wykorzystujące wspólne dane). Zależność zadań może wymuszać określoną kolejność wykonywania zadań lub konieczność ich synchronizacji. Z praktycznego punktu widzenia występowanie zadań zależnych w znacznym stopniu komplikuje zadanie szeregowania wątków/procesów w systemach czasu rzeczywistego.

Jako przykład algorytmu szeregowania zadań niezależnych, dla twardych systemów czasu rzeczywistego z jednym procesorem, można podać algorytm opublikowany przez Liu i Laylanda w 1973 roku. Algorytm wykorzystuje mechanizm wywłaszczenia i jest oparty na względnych, statycznych (niezmiennych podczas cyklu życia procesu) priorytetach i następujących założeniach:

- zgłoszenia na wykonanie wszystkich zadań posiadających twarde ograniczenia na czas reakcji są zgłoszeniami periodycznymi,
- wszystkie zadania są niezależne. Między dowolnymi dwoma zadaniami nie występują jakiekolwiek ograniczenia narzucające kolejność realizacji zadań ani konieczność wykluczenia,
- każde zadanie jest wywoływane z okresem p_i,
- maksymalny czas wykonywania każdego zadania c_i jest znany oraz niezmienny,
- czas niezbędny na przełączenie kontekstu jest pomijalnie krótki,
- maksymalny sumaryczny współczynnik obciążenia procesora $\Sigma c_i/p_i$ dla n zadań nie przekracza wartości $n(2^{1/n}-1)$. Dla $n \to \infty$ wartość ta jest równa $ln\ 2$, czyli około 0,7.

Istota algorytmu polega na tym, że wszystkim zadaniom przydziela się stałe priorytety uzależnione od czasu ich wykonywania. Zadania wykonujące się najkrócej otrzymują najwyższy priorytet, natomiast zadania wykonujące się najdłużej – najniższy. Przy spełnieniu wszystkich założeń algorytm ten gwarantuje realizację wszystkich ograniczeń dla poszczególnych zadań w dowolnej sytuacji. W przypadku gdy okresy wykonywania zadań periodycznych są wielokrotnością okresu wykonywania zadania najkrótszego, to wymaganie dotyczące maksymalnego współczynnika obciążenia procesora staje się mniej rygorystyczne i współczynnik ten może osiągać wartość 1.

Istnieją także algorytmy z dynamicznie zmienianymi priorytetami, które są przydzielane w zależności od takich bieżących parametrów poszczególnych zadań, jak np. ostateczny moment wykonania (ang. deadline). W takim przypadku planista wybiera zadanie do wykonania na podstawie bieżącej różnicy między ostatecznym momentem wykonania poszczególnych zadań i czasem niezbędnym na ich wykonanie.

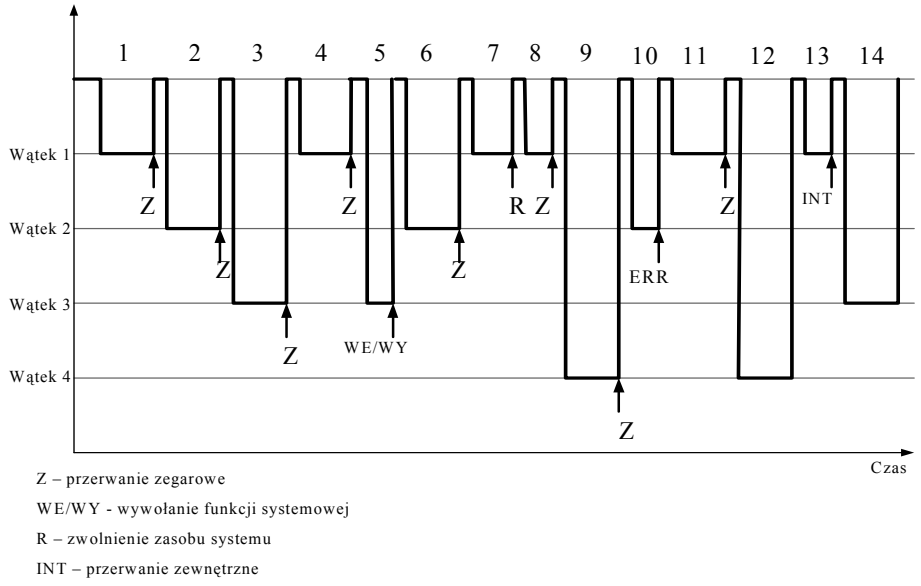
4.2.10. Uaktywnianie planisty

Podczas pracy systemu operacyjnego występują zdarzenia powodujące konieczność ponownego wykonania operacji planowania czasu procesora. Do takich zdarzeń można zaliczyć:

- wystąpienie przerwania zegarowego sygnalizującego, że czas przydzielony aktywnemu wątkowi/procesowi wyczerpał się. Planista przeprowadza wątek/proces w stan gotowości i wykonuje ponowną operację planowania,
- aktywny wątek/proces wykonał wywołanie systemowe związane z realizacją operacji WE/WY lub z dostępem do zasobu, który w danej chwili jest zajęty. Planista przeprowadza wątek/proces w stan oczekiwania i wykonuje ponowną operację planowania,
- aktywny wątek/proces wykonał wywołanie systemowe związane ze zwolnieniem zasobu. Planista sprawdza czy na ten zasób oczekuje inny wątek/proces i jeśli tak, to przeprowadza go ze stanu oczekiwania do stanu gotowości. Po wykonaniu tej operacji może się okazać, że wątek/proces, który uzyskał dostęp do zasobu, posiada wyższy priorytet od aktualnie wykonywanego. W wyniku wykonania operacji ponownego planowania następuje przydział procesora wątkowi/procesowi o najwyższym priorytecie,
- wystąpienie przerwania zewnętrznego, sygnalizującego zakończenie operacji WE/WY w urządzeniu zewnętrznym. Planista przeprowadza wątek/proces w stan gotowości i wykonuje ponowną operację planowania,
- wystąpienie przerwania wewnętrznego, sygnalizującego błąd podczas realizacji aktywnego wątku/procesu. Planista kończy wątek/proces i wykonuje ponowną operację planowania.

Po wystąpieniu każdego z powyższych zdarzeń planista przegląda kolejki i decyduje, który wątek/proces będzie wykonywany w następnej kolejności. W rzeczywistości w systemie operacyjnym występuje ponadto cały szereg innych zdarzeń (związanych przede wszystkim z wywołaniami systemowymi) wymagających ponownej operacji planowania.

Na rys. 4.14 przedstawiono przykładowy fragment diagramu czasowego pracy planisty podczas realizacji czterech wątków, a szczególną uwagę zwrócono na zdarzenia powodujące jego uaktywnienie. W pierwszych czterech krokach planista był uaktywniany przez przerwanie zegarowe (zdarzenia oznaczone literą Z). W piątym kroku jego uaktywnienie było spowodowane przez operację WE/WY w wątku trzecim. Planista przeprowadził ten wątek w stan oczekiwania a następnie przekazał procesor wątkowi nr 2, który wykonywał się przez pełen kwant czasu, zakończony przerwaniem zegarowym (krok szósty). W kroku siódmym, podczas realizacji wątku nr 1, ma miejsce wywołanie systemowe zwalniające zasób systemu (R). Planista przegląda kolejkę wątków oczekujących na zwolnienie tego zasobu i zmienia stan wątku nr 4, który oczekiwał na zwolnienie zasobu, na stan „gotowy do wykonania". Ponieważ wątek nr 4 posiada niższy priorytet od aktualnie wykonywanego wątku nr 1, planista decyduje o kontynuacji wątku nr 1 (krok ósmy). Po wyczerpaniu limitu czasu przez wątek nr 1 planista przekazuje procesor wątkowi nr 4 (krok nr 9). Podczas realizacji wątku nr 1 w kroku nr 10 ma miejsce błąd powodujący wywołanie przerwania wewnętrznego (wyjątku), co pociąga za sobą zakończenie tego wątku. W kroku nr 13 podczas realizacji wątku nr 1 występuje przerwanie zewnętrzne (INT) informujące o zakończeniu operacji WE/WY. Zdarzenie to aktywuje planistę, który przekazuje procesor do wątku nr 3, oczekującego na zakończenie operacji WE/WY, gdyż wątek nr 1 posiadał niższy priorytet.



Rys. 4.14. Przykładowy diagram czasowy pracy planisty systemu operacyjnego

4.3. Przerwania i ich obsługa

4.3.1. Typy przerwań

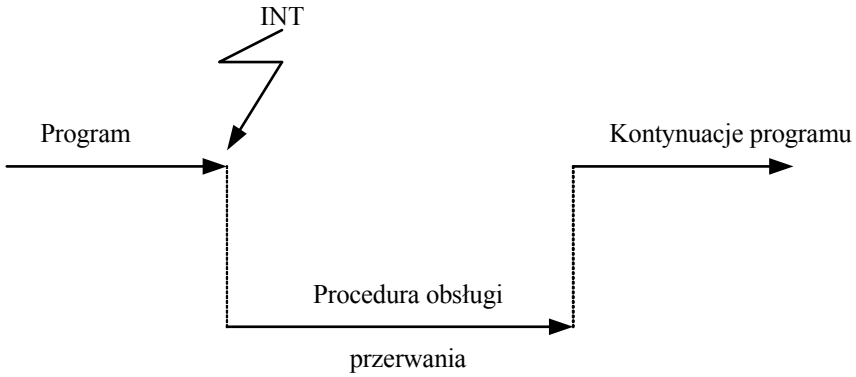
Można powiedzieć, że przerwania i ich obsługa są podstawową siłą napędową systemu operacyjnego. W przypadku gdy centralny planista wykorzystuje kwant czasu oraz mechanizm wywłaszczania, periodyczne przerwania generowane przez zegar systemowy pozwalają na przełączanie procesów, a przerwania generowane przez urządzenia WE/WY zarządzają strumieniami danych wymienianymi między jednostką centralną komputera a otoczeniem zewnętrznym.

W wyniku wystąpienia przerwania od określonego urządzenia, system przerwań komputera wstrzymuje wykonywanie przez procesor aktualnego programu i wymusza wykonanie skoku do wcześniej przygotowanego i przechowywanego w pamięci programu obsługi przerwania. Po zakończeniu wykonywania programu obsługi przerwania następuje powrót do wcześniej realizowanego programu. Przedstawiono to na rys. 4.15.

Wynika z tego, że obsługa przerwania bardzo przypomina wykonanie skoku do podprogramu. Różnica polega jednak na tym, że skoki do podprogramów występują w określonych miejscach programu, narzuconych przez programistę, natomiast przerwania mogą występować w dowolnych momentach czasu.

W zależności od źródła przerwania można wyróżnić trzy rodzaje przerwań:

- zewnętrzne (generowane przez urządzenia zewnętrzne),
- wewnętrzne, określane także mianem wyjątków (generowane wewnątrz procesora),
- programowe (będące wynikiem wykonania określonego rozkazu).



Rys. 4.15. Zasada obsługi przerwań

Przerwania zewnętrzne mogą wynikać z operacji wykonywanych przez użytkownika (klawiatura, myszka itp.) lub sygnałów generowanych przez urządzenia zewnętrzne (np. zakończenie operacji WE/WY, drukarka, dysk twardy itp.). Przerwania zewnętrzne są przekazywane do procesora poprzez wejściowy sygnał przerywający za pośrednictwem sterownika przerwań.

Przerwania wewnętrzne (wyjątki) są generowane wewnątrz procesora w wyniku błędnego wykonania rozkazu (np. operacja dzielenia przez zero, błąd ochrony pamięci, próba wykonania uprzywilejowanego rozkazu w trybie użytkownika itp.).

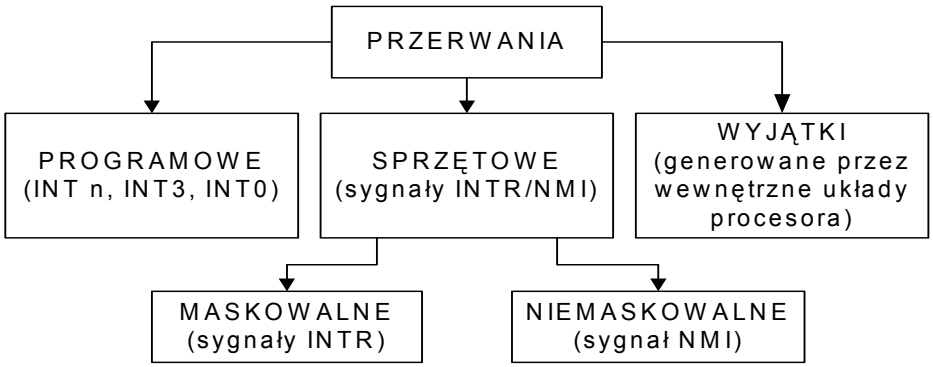
Przerwania programowe różnią się od poprzednich dwóch grup przerwań ponieważ w rzeczywistości to nie są przerwania. Przerwania programowe są generowane w wyniku wykonania określonego rozkazu, który w rzeczywistości jest rozkazem skoku do podprogramu, jednak wykonanie tego skoku jest realizowane analogicznie jak podczas obsługi przerwania.

Przerwania sprzętowe są obsługiwane przez procedury umieszczone w odpowiednich sterownikach urządzeń zewnętrznych, procedury obsługi wyjątków znajdują się w jądrze systemu operacyjnego, natomiast przerwania programowe są najczęściej obsługiwane przez procedury systemu operacyjnego (tzw. wywołania systemowe).

4.3.2. Klasyfikacja przerwań procesora Pentium w trybie chronionym

Ogólną klasyfikację przerwań przedstawiono na rys. 4.16.

Przerwania (ang. interrupts) generowane są przez urządzenia zgłaszające potrzebę obsługi (przerwania sprzętowe) lub wywoływane przez wykonywany program, w wyniku rozkazów INT n, INT3, INT0 (przerwania programowe). Z wyjątkiem przerwania niemaskowalnego, pozostałe przerwania są zgłaszane podczas normalnej pracy systemu komputerowego i stanowią podstawę obsługi urządzeń zewnętrznych oraz wywołań funkcji systemowych. Procesory Pentium udostępniają jedno przerwanie niemaskowalne o najwyższym priorytecie. Z tego powodu jest ono wykorzystywane w systemach komputerowych do obsługi zdarzeń ekstremalnych (np. błąd pamięci, zanik napięcia zasilania). *Wyjątki* natomiast generowane są przez wewnętrzne układy procesora, w wyniku błędów występujących podczas wykonywania programu.



Rys. 4.16. Klasyfikacja przerwań i wyjątków w trybie chronionym procesora Pentium

Procesory Pentium (tak w trybie rzeczywistym jak i chronionym) obsługują maksymalnie 256 przerwań wszystkich typów (przerwań i wyjątków). W trybie

chronionym rozszerzono listę wyjątków do 32 (numery 0...31) z ośmiu w trybie rzeczywistym (numery 0...7), przy czym w aktualnych procesorach Pentium wykorzystywanych jest 19 pierwszych (pozostałe są zarezerwowane). Wyjątki można podzielić na trzy grupy: błędy, pułapki oraz załamania.

Błąd (ang. error) jest wykrywany i obsługiwany przed wykonaniem rozkazu, który jest przyczyną jego wystąpienia. Przykładem może być sytuacja mająca miejsce podczas odwołania się kolejnego rozkazu do strony lub segmentu nieobecnego aktualnie w pamięci. Adresem powrotu do programu, po wykonaniu obsługi błędu, jest adres rozkazu będącego przyczyną generacji wyjątku.

Pułapka (ang. trap) to wyjątek, który jest generowany po wykonaniu rozkazu. Przykładem może być rozkaz INT0, wywołujący wyjątek o numerze 4 w przypadku, gdy w rejestrze EFLAGS znacznik OF=1 (przepełnienie – przekroczenie zakresu liczb ze znakiem w kodzie U2). Adresem powrotu do programu, po wykonaniu obsługi błędu, jest adres rozkazu znajdującego się bezpośrednio za rozkazem będącym przyczyną generacji wyjątku. Po wystąpieniu błędu czy pułapki oraz po wykonaniu programu obsługi danego wyjątku program jest nadal wykonywany.

Załamanie (ang. abort) to wyjątek generowany w sytuacji, gdy procesor nie może dalej realizować programu. Przykładem może być wyjątek nr 8 generowany w sytuacji, gdy podczas przekazywania sterowania procedurze obsługi pewnego wyjątku, pojawia się inny wyjątek (tzw. wyjątek podwójnego błędu), uniemożliwiający prawidłowe zakończenie procesu. W takim przypadku procesor generuje sygnał RESET.

W tabeli 4.1 zebrano wszystkie wyjątki procesorów rodziny Pentium.

Tabela 4.1 Wyjątki procesorów rodziny Pentium

Numer przerwania	Oznaczenie	Opis	Rodzaj wyjątku
0	#DE	Błąd dzielenia przez zero	Błąd
1	#DB	Praca krokowa	Błąd/Pułapka
2		Przerwanie niemaskowalne NMI	Przerwanie
3	#BP	Rozkaz INT3, realizacja pułapek przez debugery	Pułapka
4	#OE	Wyjątek generowany w przypadku, gdy podczas wykonania rozkazu INT0 jest ustawiony znacznik nadmiaru	Pułapka
5	#BR	Wyjątek generowany w przypadku, gdy adres względny w segmencie (OFFSET) przekracza granicę segmentu	Błąd

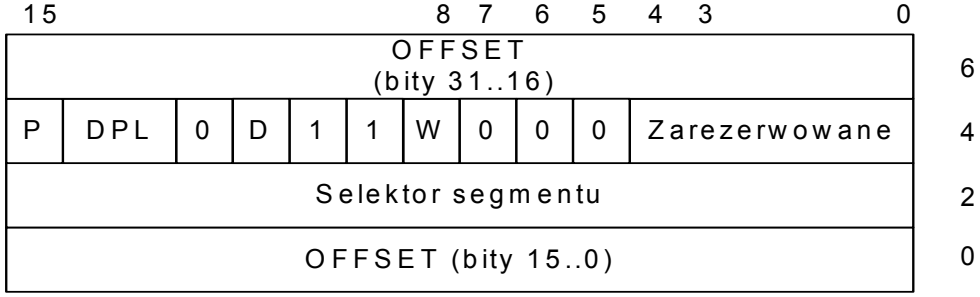
6	#UD	Niedozwolony kod rozkazu	Błąd
7	#NM	Niedostępna zmiennoprzecinkowa jednostka arytmetyczna	Błąd
8	#DF	Podwójny błąd	Załamanie
9		Zarezerwowany	
10	#TS	Wyjątek generowany w przypadku, gdy podczas przełączania zadania wystąpił błąd związany z operacjami na segmencie stanu zadania TSS	Błąd
11	#NP	Brak segmentu (procesor odwołuje się do deskryptora segmentu, który aktualnie nie jest w pamięci)	Błąd
12	#SS	Wyjątek generowany w przypadku błędnych operacji w segmencie stosu (np. segment stosu nieobecny w pamięci lub przekroczenie granicy stosu)	Błąd
13	#GP	Ogólne naruszenie mechanizmu ochrony	Błąd
14	#PF	Błąd stronicowania	Błąd
15		Zarezerwowany	
16	#MF	Błąd zmiennoprzecinkowej jednostki arytmetycznej	Błąd
17	#AC	Niewyrównany adres argumentu w pamięci	Błąd
18	#MC	Kontrola maszynowa. Wyjątek generowany w sytuacji wystąpienia awarii sprzętowych (np. magistrali, pamięci CACHE)	Załamanie
19	#XM	Wyjątek generowany przez SSE	Błąd
20-31		Zarezerwowane	
32-255		Przerwania	

4.3.3. Obsługa przerwań w trybie chronionym procesora Pentium

W trybie chronionym wywołanie podprogramów obsługujących przerwania odbywa się nie poprzez tablicę wektorów przerwań umieszczoną pod adresem zerowym (obowiązującą dla trybu rzeczywistego), lecz poprzez tablicę deskryptorów przerwań IDT (ang. Interrupt Descriptor Table), która może być umieszczona w dowolnym miejscu przestrzeni adresowej. Adres bazowy tablicy IDT przechowywany jest w rejestrze IDTR. Załadowanie rejestru IDTR odbywa się przy pomocy instrukcji LIDT, która może być wykonana w programie o najwyższym poziomie przywileju CPL=0, natomiast odczyt – przy pomocy instrukcji SIDT, która może się znaleźć w dowolnym programie wykonywanym w trybie chronionym.

Elementami tablicy IDT są 8-bajtowe deskryptory przerwań, co przy liczbie przerwań równej 256 daje rozmiar tablicy 2 kB. W tablicy IDT mogą znajdować się trzy typy deskryptorów systemowych: furtki przerwania (ang. interrupt gate), furtki pułapki (ang. trap gate) oraz furtki zadania (ang. task gate). Struktura deskryptora furtki przerwania oraz pułapki została przedstawiona na rys. 4.17.

Deskryptory te zawierają przede wszystkim selektor segmentu oraz adres względny (OFFSET) w tym segmencie, pod którym rozpoczyna się procedura obsługi przerwania. Poza tym deskryptor zawiera następujące pola: bit P określający czy segment jest dostępny w pamięci (P=1), dwubitowe pole DPL określające poziom przywileju furtki, bit D określający tryb pracy procesora przy obsłudze przerwań (dla D=0 tryb 16-bitowy procesorów 80286, dla D=1 tryb 32-bitowy procesorów 80386 oraz nowszych), a także bit W określający czy deskryptor dotyczy furtki przerwania (W=0), czy furtki pułapki (W=1).



Rys. 4.17. Format deskryptora przerwania oraz pułapki

Skok do procedury obsługi przerwania lub pułapki z wykorzystaniem odpowiedniej furtki może być związany ze zmianą poziomu przywileju. Procesor sprawdza wartość pola DPL w deskryptorze furtki podczas wykonywania instrukcji przerwań programowych INT n, INT3 oraz INT0. Skok do procedury obsługi jest wykonywany jedynie w przypadku, gdy wartość wpisana w polu poziomu przywileju programu wykonującego powyższe

rozkazy jest mniejsza od wartości wpisanej do poziomu przywileju furtki: $DPL_{progr}<=DPL_{furtki}$ (najwyższy poziom priorytetu określony jest przez wartość 0). W ten sposób DPL_{furtki} określa jakie programy mogą wywoływać podane wyżej przerwania programowe. Podczas obsługi innych przerwań oraz wyjątków procesor ignoruje pole DPL_{furtki}.

Poza tym poziom przywileju segmentu, w którym jest umieszczona procedura obsługi przerwania, powinien być mniejszy lub równy poziomowi przywileju segmentu, w którym znajduje się bieżący program. Naruszenie tej zasady powoduje generację wyjątku ogólnego naruszenia mechanizmu ochrony. Z tego powodu zalecane jest, aby procedury obsługi przerwań i wyjątków znajdowały się w segmencie o poziomie przywileju DPL=0. W ten sposób unika się możliwości wywoływania przerwań programowych przez programy o niższym poziomie przywileju.

Po wystąpieniu przerwania procesor odkłada na stos następujące dane (rys. 4.18):

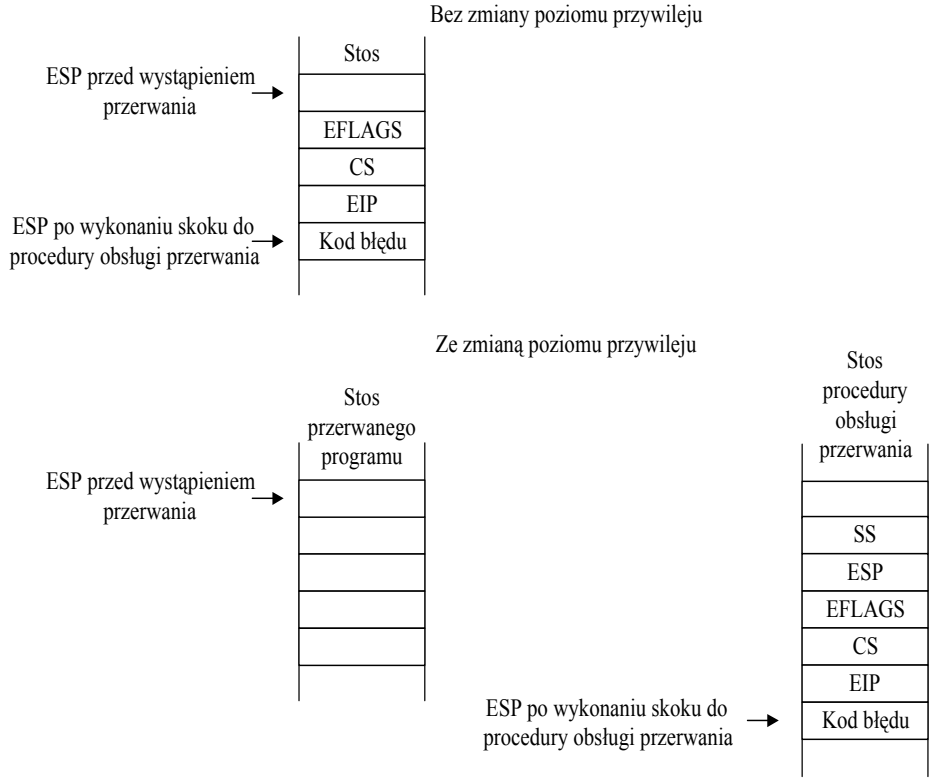
- zawartość rejestru SS przerwanego programu (tylko w przypadku, gdy ma miejsce zmiana poziomu przywileju),
- zawartość rejestru ESP przerwanego programu (tylko w przypadku, gdy ma miejsce zmiana poziomu przywileju),
- zawartość rejestru EFLAGS,
- zawartość rejestru CS,
- zawartość rejestru EIP,
- kod błędu (tylko w przypadku wystąpienia wyjątku).

W przypadku gdy poziom przywileju procedury obsługi przerwania/wyjątku jest taki sam jak poziom przywileju przerwanego programu, podczas obsługi przerwania wykorzystywany jest stos przerwanego programu. Jeżeli natomiast obsługa przerwania/wyjątku związana jest ze zmianą poziomu przywileju, stos przerwanego programu nie jest wykorzystywany, natomiast poszczególne dane odkładane są na stos w procedurze obsługi przerwania. Poza tym obsługa przerwania w trybie chronionym może być wykonana poprzez operację przełączenia zadania.

4.3.4. Zarządzanie przerwaniami w systemach operacyjnych

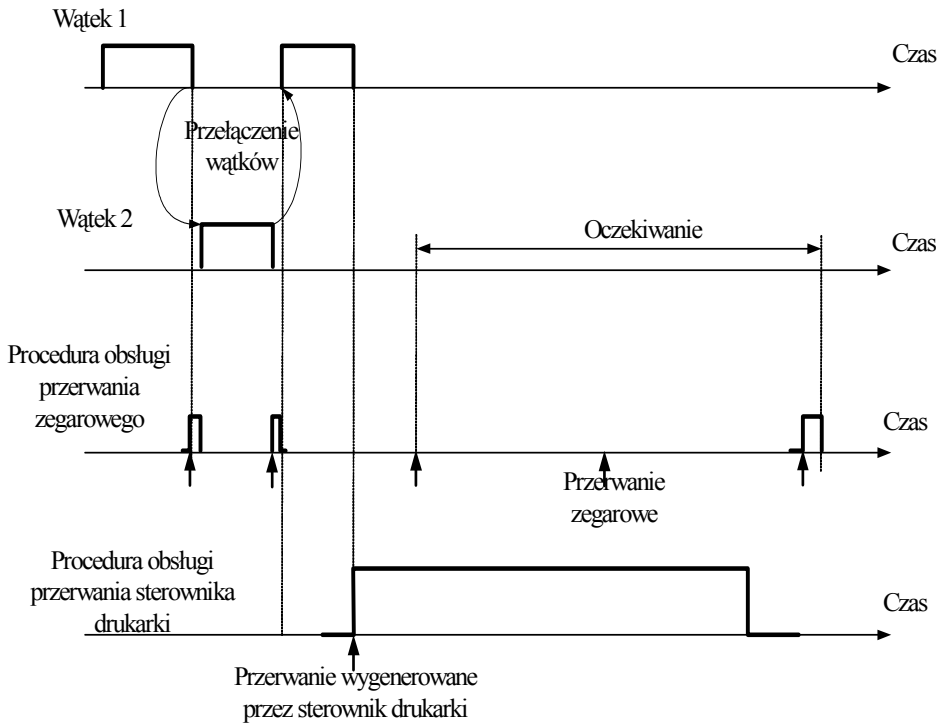
Przerwania odgrywają bardzo ważną rolę w funkcjonowaniu systemu komputerowego, gdyż umożliwiają reakcję na zdarzenia asynchroniczne względem pracy systemu. System operacyjny musi odgrywać aktywną rolę w organizacji obsługi przerwań. Główna trudność w obsłudze przerwań wynika przede wszystkim z niedających się przewidzieć momentów skoków do różnych procedur, spowodowanych przez zdarzenia zewnętrzne lub wewnętrzne. System operacyjny nie może stracić kontroli nad wykonywaniem procedur systemowych przeprowadzanych w ramach obsługi przerwań. W tym celu powinien szeregować w czasie wszystkie przerwania, analogicznie do szeregowania wątków w systemie operacyjnym. Należy pamiętać, że planista

systemu operacyjnego jest także procedurą systemową wywoływaną przez przerwania (sprzętowe – generowane przez zegar systemowy lub kontroler urządzenia WE/WY, programowe – generowane przez programy użytkowe oraz inne moduły jądra). Z tego powodu prawidłowe planowanie procedur wywoływanych przez przerwania jest koniecznym warunkiem prawidłowego planowania wątków użytkowników.



Rys. 4.18. Wykorzystanie stosu podczas obsługi przerwań i wyjątków w trybie chronionym procesora Pentium

Przykład nieprawidłowej obsługi przerwań przedstawiono na rys. 4.19, w którym program obsługi drukarki blokuje na długi okres obsługę przerwania zegarowego, na skutek czego czas systemowy na długo „zamiera", a wątek nr 2 musi oczekiwać na zakończenie operacji związanych z obsługą przerwania zgłoszonego przez sterownik drukarki.



Rys. 4.19. Przykład nieprawidłowej obsługi przerwań

Obsługa przerwań w systemach operacyjnych odbywa się na podstawie kolejek priorytetowych. Wszystkie źródła przerwań są zwykle dzielone na kilka klas o różnych priorytetach, a problemem szeregowania przerwań zajmuje się moduł systemu operacyjnego określany mianem zarządcy przerwań, który jest wywoływany przez każde przerwanie. Program ten na krótko blokuje przerwania, a w tym czasie wykonuje następujące operacje:

- określa źródło przerwania,
- sprawdza, czy procesor aktualnie obsługuje inne przerwanie,
- jeżeli tak, to porównuje priorytet obsługiwanego przerwania z priorytetem zgłoszonego przerwania,
- jeżeli priorytet nowego przerwania jest wyższy od priorytetu przerwania aktualnie obsługiwanego, to następuje zawieszenie obsługi tego przerwania i wykonanie obsługi nowego,
- w przeciwnym wypadku wstawia nowe przerwanie do priorytetowej kolejki przerwań.

4.3.5. Zarządzanie przerwaniami w systemie Windows NT

Wszystkie źródła przerwań (sprzętowych, programowych oraz wyjątków) są podzielone na kilka klas priorytetowych, a każdej klasie zostaje przydzielony priorytet, określany mianem IRQL (ang. Interrupt Request Level). Za obsługę

przerwań jest odpowiedzialny programowy zarządca przerwań w systemie Windows NT (ang. Trap Handler). Po wystąpieniu dowolnego przerwania zostaje wywołany zarządca przerwań, który zapamiętuje źródło przerwania oraz określa jego priorytet. Jeśli priorytet zgłoszonego przerwania jest mniejszy lub równy IRQL aktualnie wykonywanego programu, zgłoszenie zostaje umieszczone w kolejce zgłoszeń, po czym wykonywany jest powrót do przerwanego programu. Po zakończeniu tego programu następuje powrót do zarządcy przerwań, który analizuje kolejkę zgłoszonych przerwań i wybiera do obsługi przerwanie o najwyższym priorytecie. W przypadku gdy zgłoszone przerwanie posiada priorytet wyższy od IRQL aktualnie wykonywanego programu, następuje jego wywłaszczenie i ustawienie do kolejki związanej z danym poziomem IRQL, a sterowanie zostaje przekazane do programu obsługi zgłoszonego przerwania.

W Windows NT najniższy poziom IRQL odpowiada wątkom zarządzanym przez zarządcę wątków (wątki procesów użytkowych) (tabela 4.2), natomiast najwyższy – procedurom obsługi zdarzeń „dramatycznych", jak błąd szyny i podobne awarie sprzętowe oraz zanik napięcia zasilania czy wywołanie przerwania międzyprocesorowego. Szczególną rolę odgrywa przerwanie zegarowe posiadające bardzo wysoki priorytet. W oparciu o to przerwanie system operacyjny szereguje wątki/procesy z wykorzystaniem kwantu czasu.

Przerwania zgłaszane przez urządzenia zewnętrzne zajmują środkowe poziomy IRQL. Wzajemne związki priorytetów przerwań zewnętrznych są określone przez sprzętowy system obsługi przerwań systemu komputerowego.

Do obsługi przerwań programowych przeznaczono dwa poziomy priorytetów IRQL. Przerwania programowe obsługujące wywołania systemowe wykonane z aplikacji posiadają niższy poziom priorytetu (APC – ang. Asynchronous Procedure Call) niż przerwania programowe wywoływane z modułów jądra systemu operacyjnego (DPC – ang. Dispacher Procedure Call). Do kolejki DPC wprowadzane są wywołania programowe odwołujące się do zarządcy wątków, związane z planowaniem wątków.

Tabela 4.2. Poziomy IRQL

Najwyższy (błąd szyny, ...)
Zanik zasilania
Przerwanie międzyprocesowe
Zegar
Urządzenie nr n
...
Urządzenie nr 1
DPC
APC
Wątki użytkowe

4.3.6. Wywołania systemowe

Wywołania systemowe pozwalają aplikacjom zwracać się do systemu operacyjnego z prośbą o wykonanie określonej usługi. Dla programisty programów użytkowych system operacyjny jest biblioteką procedur/funkcji, przy pomocy których można wykonać szereg operacji, wykonanie których byłoby niemożliwe w trybie użytkownika (np. wymiana danych z urządzeniem WE/WY). Realizacja wywołań systemowych powinna spełniać następujące wymagania:

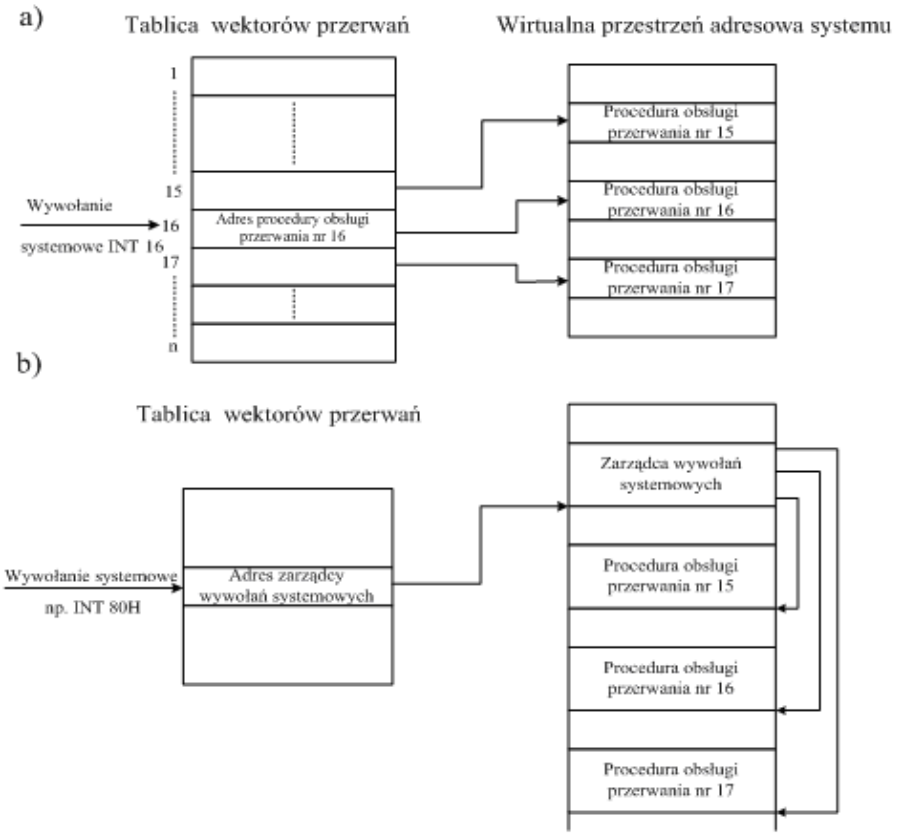
- zapewniać przełączenie w tryb jądra (uprzywilejowany),
- zapewniać dużą szybkość wywoływania procedur systemu operacyjnego,
- w miarę możliwości zapewnić jednakowe odwołania do procedur systemowych niezależnie od platformy sprzętowej, na której pracuje system operacyjny,
- umożliwiać łatwe rozszerzanie zbioru wywołań systemowych,
- zapewnić kontrolę prawidłowości wywołań systemowych ze strony systemu operacyjnego.

Dla większości platform sprzętowych spełnienie pierwszego wymagania jest możliwe jedynie przy zastosowaniu przerwań programowych i dlatego pozostałe wymagania muszą być spełnione dla takiej realizacji wywołań systemowych. Przerwanie programowe realizuje jeden ze sposobów skoku do podprogramu przy pomocy specjalnego rozkazu (INT w procesorze Pentium, TRAP w mikroprocesorach Motorola itd.). Rozkaz przerwania programowego jest rozkazem skoku pośredniego do podprogramu, w którym zamiast adresu podprogramu podaje się numer pozycji w tablicy wektorów przerwań, w której znajduje się adres procedury.

Wywołania procedur systemowych mogą być realizowane w dwojaki sposób: scentralizowany oraz zdecentralizowany (rys. 4.20). W realizacji zdecentralizowanej poszczególne wektory przerwań zawierają adresy odpowiadających im procedur systemowych (rys. 4.20a). Rozwiązanie takie posiada istotne wady, a mianowicie zależność od konkretnej platformy sprzętowej oraz trudność w rozbudowie, a także modyfikacji wywołań systemowych. Ponadto przy ograniczonej liczbie pozycji w tablicy wektorów przerwań (dla procesora Pentium – 256) w systemie obsługiwana może być niewielka liczba wywołań systemowych. Zdecentralizowaną obsługę wywołań systemowych dla wywołań usług BIOS stosowano w komputerach PC w systemie DOS.

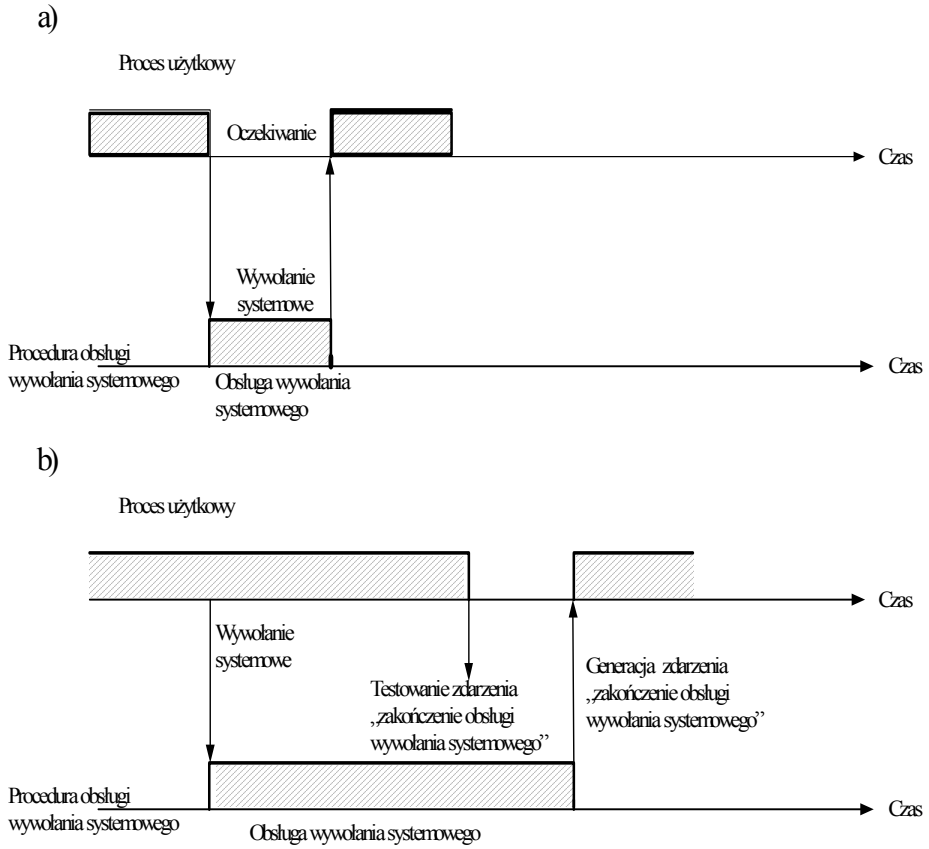
W większości współczesnych systemów operacyjnych stosowana jest scentralizowana obsługa wywołań systemowych (rys. 4.20b). W takim przypadku wszystkie wywołania systemowe są wykonywane w taki sam sposób (np. w systemie DOS – INT 21H, w systemie Linux – INT 80H, w Windows NT – INT 2EH). Powoduje to wywołanie programu zarządcy wywołań systemowych, który z kolei – na podstawie przekazanej informacji o numerze

konkretnej procedury - wywołuje odpowiednią procedurę obsługi. Przekazanie do zarządcy numeru procedury oraz ewentualnych parametrów może odbywać się poprzez rejestry robocze procesora lub poprzez stos.



Rys. 4.20. Zdecentralizowana oraz scentralizowana obsługa wywołań systemowych

Wywołania systemowe mogą być wykonywane przez system operacyjny w trybie synchronicznym i asynchronicznym. Synchroniczne wywołanie systemowe oznacza, że po wykonaniu wywołania proces zostaje zatrzymany (przeniesiony przez planistę w stan oczekiwania) aż do momentu zakończenia procedury obsługi wywołania systemowego, po czym zostaje przeniesiony w stan gotowości (rys. 4.21a). Asynchroniczne wywołanie systemowe nie wykonuje blokady procesu (rys. 4.21b).



Rys. 4.21. Synchroniczne i asynchroniczne wywołania systemowe

4.4. Synchronizacja wątków/procesów

W wielozadaniowych systemach operacyjnych istnieje konieczność synchronizacji wątków/procesów zależnych, czyli wątków/procesów jednocześnie wykorzystujących wspólne zasoby (sprzętowe oraz informacyjne) systemu komputerowego. Synchronizacja jest niezbędna w celu eliminacji wyścigów oraz blokad między wątkami podczas jednoczesnego dostępu do wspólnych urządzeń WE/WY oraz danych. W wielu systemach operacyjnych środki takie określa się mianem środków komunikacji międzyprocesowej (ang. IPC – Inter Process Communications), co jest znacznie szerszym pojęciem, zawierającym nie tylko środki synchronizacji wątków, lecz także środki wymiany danych między wątkami. Wynika to z faktu, że szereg mechanizmów wymiany danych między wątkami/procesami realizuje jednocześnie operacje związane z synchronizacją.

Wykonywanie wątku w środowisku wielozadaniowym odbywa się zawsze asynchronicznie. Wynika to z faktu, że nie jest możliwe określenie przez

programistę czy użytkownika w jakim momencie nastąpi przełączenie kontekstu i przekazanie sterowania do innego wątku/procesu. Nie jest także możliwe określenie na etapie tworzenia programu w jakim stanie (podczas realizacji programu) będą kolejki do poszczególnych zasobów systemu komputerowego, co nie pozwala określić kolejności realizacji poszczególnych wątków/procesów. Ponieważ w momencie uruchomienia danego programu w systemie operacyjnym może funkcjonować dowolny zestaw wątków (wynikający z dotychczasowej pracy systemu), więc programista może jedynie założyć, że kolejność realizacji wątków oraz momenty ich przełączania są przypadkowe.

Jeśli wątki/procesy są niezależnie (nie wykorzystują wspólnych zasobów komputera), kolejność ich realizacji w systemie operacyjnym jest nieistotna, a programista tworzący program nie musi brać pod uwagę względnego stanu realizacji poszczególnych wątków. Innymi słowy, programista tworzący program niekorzystający ze wspólnych danych, tworzy go tak, jakby miał on być wykonywany w środowisku jednozadaniowym. Jeśli jednak tworzony program przetwarza dane, które będą jednocześnie przetwarzane przez inne wątki/procesy, wykonywane współbieżnie w systemie operacyjnym, to z zasady jest istotne czy określoną modyfikację wspólnych danych wykona w pierwszej kolejności wątek/proces *A* a następnie wątek/proces *B*, czy odwrotnie. W takim przypadku staje się niezbędna operacja synchronizacji wątków/procesów.

Synchronizacja wątków/procesów jest związana z koordynacją szybkości ich realizacji i polega na wstrzymywaniu realizacji jednego (lub więcej) z nich do momentu wystąpienia określonego zdarzenia. Np. wątek pobierający dane z bufora musi oczekiwać tak długo, jak długo bufor będzie pusty. Jeśli inny wątek wstawi określone dane do bufora, to dany wątek może je pobrać i kontynuować działanie. Z drugiej strony, jeśli bufor został zapełniony, wątek wstawiający dane do bufora musi oczekiwać aż inny wątek pobierze dane i zwolni miejsce w buforze.

Analogicznie synchronizacja wątków jest niezbędna w przypadku jednoczesnego dostępu wątków do określonego zasobu sprzętowego. Jeśli np. aktywny wątek potrzebuje dostępu do portu szeregowego, a w tym momencie port jest wykorzystywany przez inny wątek, to system operacyjny wstrzymuje aktywny wątek do momentu, aż port szeregowy zostanie zwolniony.

W celu synchronizacji wątków wchodzących w skład określonej aplikacji można wykorzystywać mechanizmy systemowe, lecz także zmienne globalne dostępne dla wszystkich wątków danej aplikacji. Jednak metoda ta może okazać się zawodna, a w przypadku synchronizacji wątków wchodzących w skład różnych procesów – wprost niemożliwa, gdyż wątki nie posiadają dostępu do przestrzeni adresowych innych procesów. Z tego powodu podczas synchronizacji wątków praktycznie jest niezbędne wykorzystywanie mechanizmów systemowych. Z zasady twórcy systemów operacyjnych udostępniają szeroki wachlarz środków synchronizacji wątków. Środki te mogą tworzyć pewną hierarchię, ponieważ jedne z nich (umieszczone na wyższym

poziomie hierarchii) wykorzystują prostsze środki synchronizacji. Poza tym pewne środki synchronizacji mogą być przeznaczone wyłącznie do synchronizacji wątków tego samego procesu, podczas gdy inne mogą być stosowane dla wątków różnych procesów. Z zasady programista ma do wyboru szereg środków, z których może korzystać zamiennie.

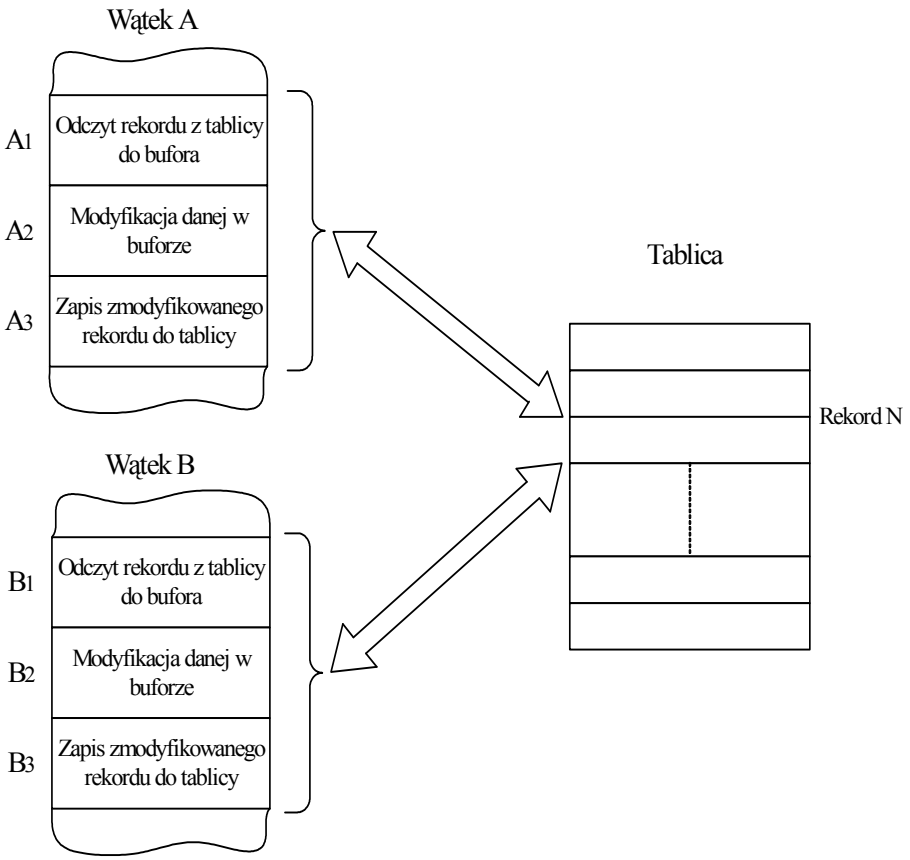
4.4.1. Konieczność synchronizacji wątków oraz pojęcie wyścigu

Zaniedbanie problemów synchronizacji w systemie wielowątkowym może doprowadzić do nieprawidłowego wykonywania operacji. Rozpatrzmy zadanie jednoczesnego wykonywania operacji na danych przechowywanych w pamięci przez dwa (lub więcej) wątki. Załóżmy, że w pamięci przechowywana jest tablica zawierająca dane o klientach (indeksowane numerem klienta) oraz dane o aktualnych stanach kont (rys. 4.22). W programie przetwarzającym dane mogą być jednocześnie uruchomione dwa (lub więcej) wątki modyfikujące stany kont. Wszystkie wątki wykonują analogiczne operacje:

1. odczyt stanu konta określonego klienta z tablicy do bufora roboczego,
2. modyfikacja stanu konta w buforze,
3. zapis zmodyfikowanego stanu konta do tablicy.

Załóżmy, że w pewnej chwili wątek *A* rozpoczyna operację modyfikacji stanu konta klienta o numerze *N*. W tym celu wątek odczytuje określone dane z tablicy do swego bufora roboczego (krok A_1), modyfikuje dane w buforze (krok A_2), lecz nie zapisuje zmodyfikowanych danych do tablicy, gdyż system operacyjny wstrzymuje jego wykonywanie (wywłaszczenie spowodowane np. przez wyczerpanie kwantu czasu).

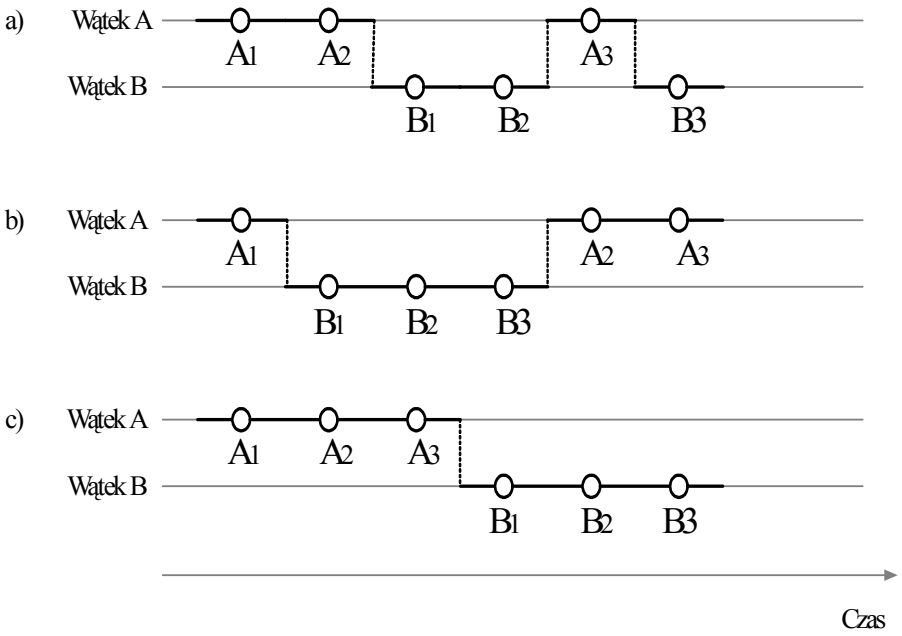
Załóżmy następnie, że wątek *B* rozpoczyna operację modyfikacji stanu konta tego samego klienta. W tym celu wątek *B* odczytuje dane z tablicy do swojego bufora roboczego (niezmodyfikowane jeszcze przez wątek *A*), modyfikuje stan konta, a następnie zostaje wywłaszczony analogicznie jak wątek *A*. Po pewnym czasie system operacyjny decyduje o kontynuacji wątku *A*, co powoduje zapis do tablicy danych zmodyfikowanych przez ten wątek, a następnie – po zakończeniu operacji przez wątek *A* – kontynuuje wątek *B* i zapisuje do tablicy stan konta zmodyfikowany w tym wątku. W wyniku takiej sekwencji zdarzeń w tablicy zapisana jest wyłącznie zmiana stanu konta dokonanego w wątku *B*, podczas gdy modyfikacja dokonana w wątku *A* została zniszczona (nadpisana).



Rys. 4.22. Wystąpienie wyścigów podczas modyfikacji wspólnych danych przez wiele wątków

Podczas wykonywania analogicznych operacji przez wątki *A* oraz *B* mogą wystąpić różne sekwencje zdarzeń, które powodują nieprawidłowe zapisy danych w tablicy (rys. 4.23). Jedynie sekwencje zdarzeń, w których najpierw zostaną wykonane wszystkie trzy kroki wątku *A,* a następnie wątku *B* (lub odwrotnie), dadzą poprawne wyniki.

Takie sytuacje, w których dwa (lub więcej) wątki modyfikują wspólne dane, a końcowy rezultat zależy od sekwencji realizacji wątków (od ich wzajemnej szybkości realizacji), określane są mianem *wyścigów*.



Rys. 4.23. Zależność wyniku od względnej szybkości realizacji wątków

4.4.2. Sekcja krytyczna

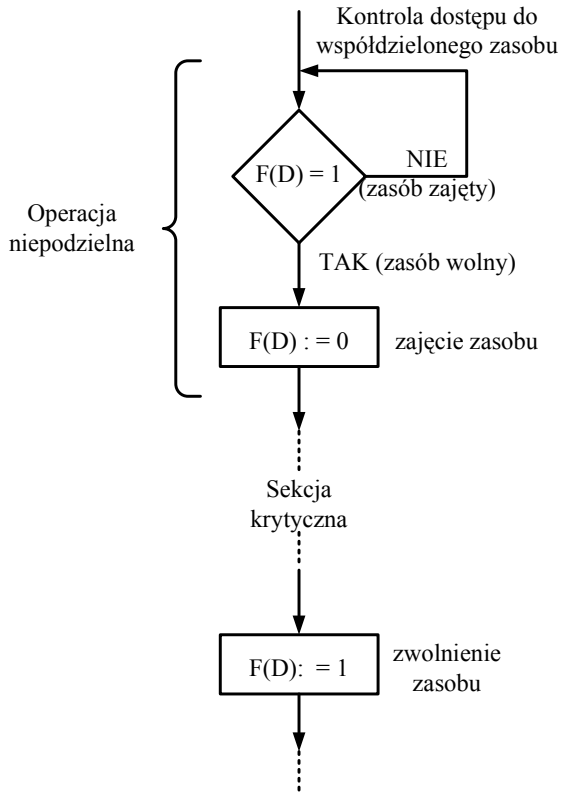
Zasadniczym pojęciem, ważnym dla wyjaśnienia problemów synchronizacji wątków, jest pojęcie sekcji krytycznej. Sekcja krytyczna zawsze jest związana z określonymi krytycznymi danymi, które mogą być modyfikowane jednocześnie przez wiele wątków. Można powiedzieć, że w skład sekcji krytycznej danego wątku wchodzi sekwencja operacji modyfikacji danych, które mogą być jednocześnie modyfikowane w innych wątkach. W celu eliminacji wyścigów, które mogą mieć miejsce podczas modyfikacji tych samych danych przez wiele współbieżnie wykonywanych wątków, należy zapewnić, aby w każdej chwili w sekcji krytycznej związanej z określonymi danymi krytycznymi, znajdował się wyłącznie jeden wątek, niezależnie od tego w jakim jest stanie (np. aktywny lub gotowy do wykonania). Operację tę określa się mianem wzajemnego wykluczania.

4.4.3. Zmienne blokujące

W celu synchronizacji wątków należących do jednego procesu programista może używać globalnych zmiennych, dostępnych dla wszystkich wątków, które mogą pełnić rolę *zmiennych blokujących*. W tym celu poszczególne dane krytyczne programista wiąże z odpowiadającymi im zmiennymi logicznymi. Wątek przypisuje danej zmiennej blokującej wartość 0 – gdy wchodzi do

odpowiedniej sekcji krytycznej, oraz wartość 1 – gdy opuszcza sekcję krytyczną.

Na rys. 4.24 przedstawiono fragment algorytmu realizowanego przez wątek wykorzystujący zmienną blokującą *F* do wzajemnego wykluczania dostępu wątków do danych krytycznych *D*, co opisujemy zależnością *F(D)*. Przed wejściem do sekcji krytycznej wątek sprawdza czy inny wątek nie znajduje się w analogicznej sekcji krytycznej. W tym celu testuje wartość zmiennej *F*, a następnie – jeżeli zmienna zawiera wartość 1 – wpisuje do niej wartość 0 i rozpoczyna realizację sekcji krytycznej. W przypadku gdy zmienna *F* posiada wartość 0 (co oznacza, że inny wątek znajduje się aktualnie w sekcji krytycznej związanej z tymi samymi zmiennymi krytycznymi), wątek realizuje cykliczne sprawdzanie wartości zmiennej *F*. Po zakończeniu realizacji instrukcji wchodzących w skład sekcji krytycznej wątek wpisuje do zmiennej blokującej *F* wartość 1, co pozwala innym wątkom na wejście do odpowiedniej sekcji krytycznej.



Rys. 4.24. Realizacja sekcji krytycznej z zastosowaniem zmiennych blokujących

Przedstawiony algorytm funkcjonuje prawidłowo, jednak posiada kilka istotnych wad powodujących, że praktyczne jego zastosowanie jest niemożliwe. Po pierwsze należy pamiętać, że wszystkie wątki danego procesu

wykorzystujące zmienną blokującą *F* są wykonywane współbieżnie, co oznacza, że w dowolnej chwili system operacyjny może wywłaszczyć jeden z wątków i procesor przekazać innemu. Zastanówmy się nad tym co się stanie, gdy operacja przełączenia kontekstu wystąpi między operacją testowania zmiennej *F* a wpisaniem do tej zmiennej wartości 0. Przyjmijmy, że w wyniku testowania wartości zmiennej *F* wątek stwierdził, że określony zasób jest wolny (*F=1*) i przed operacją wpisania do zmiennej *F* wartości 0 następuje wywłaszczenie wątku. Procesor zostaje przekazany drugiemu wątkowi, posiadającemu sekcję krytyczną związaną z tym samym zasobem, który po sprawdzeniu wartości zmiennej blokującej *F* wpisuje do niej wartość 0 i rozpoczyna realizację sekcji krytycznej. Załóżmy następnie, że przed zakończeniem sekcji krytycznej następuje wywłaszczenie wątku i przekazanie procesora do pierwszego z wątków, który wpisuje teraz do zmiennej blokującej *F* wartość 0 i rozpoczyna realizację swojej sekcji krytycznej. Wynika z tego, że została naruszona zasada wzajemnego wykluczania i obydwa wątki realizują jednocześnie swoje sekcje krytyczne związane z tymi samymi zmiennymi krytycznymi.

Z przedstawionego przykładu wynika, że w celu wzajemnego wykluczania wątków należy zapewnić, aby operacja testowania wartości zmiennej blokującej *F* oraz wpisania do tej zmiennej wartości 0 była niepodzielna. Może to być zrealizowane w dwojaki sposób:

- obydwie operacje są realizowane przez jeden rozkaz procesora (np. dla procesora Pentium rozkazy BTC, BTR oraz BTS),
- obydwie operacje są wykonywane przez funkcję systemową, a początek sekcji krytycznej jest w programie określony przez wywołanie odpowiedniej funkcji systemowej.

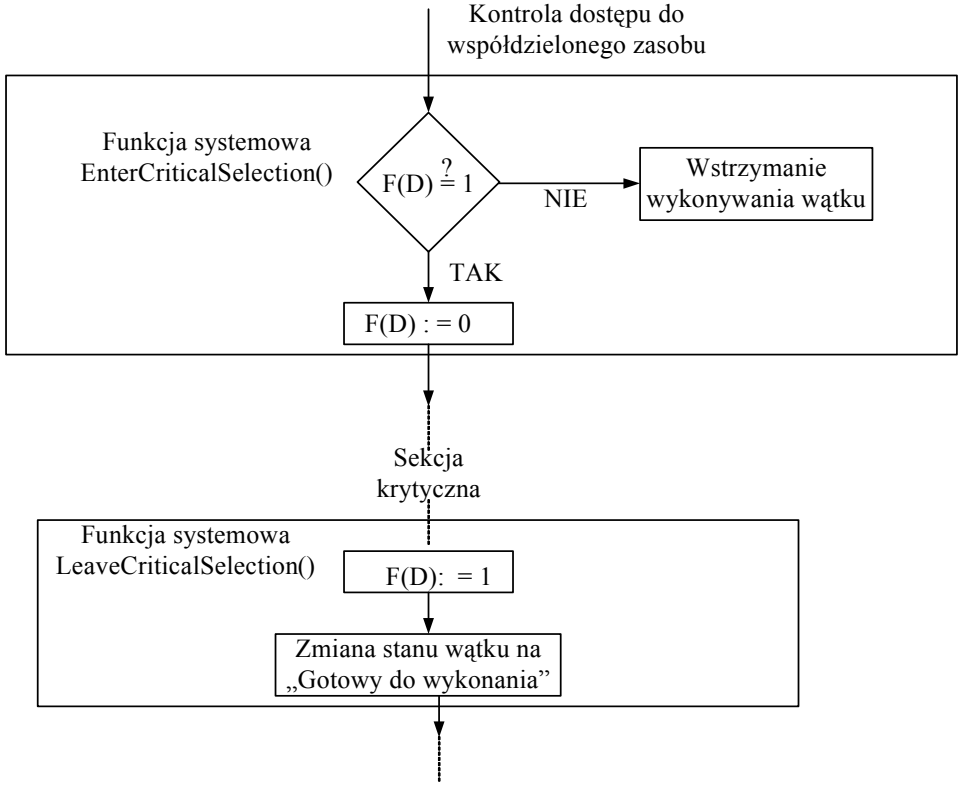
Drugą istotną wadą algorytmu przedstawionego na rys. 4.24 jest fakt, że wątek oczekujący na wejście do sekcji krytycznej wykonuje cykliczną operację testowania zmiennej blokującej, co w znacznym stopniu obciąża procesor. Aby tego uniknąć, na początku sekcji krytycznej powinna być wykonana odpowiednia funkcja systemowa, która sprawdzi stan zmiennej blokującej i jeżeli jej wartość jest równa 0 wstrzyma wykonywanie wątku (na rys. 4.25 funkcja `EnterCriticalSection()`). Analogicznie na końcu sekcji krytycznej wykonana zostaje odpowiednia funkcja systemowa wpisująca do zmiennej blokującej wartość 1 oraz, jeżeli na danej zmiennej blokującej oczekuje inny wątek, przeprowadza zmianę stanu wątku na „gotowy do wykonania” (na rys. 4.25 funkcja `LeaveCriticalSection()`).

4.4.4. Semafory

Semafory są pewnym uogólnieniem zmiennych blokujących. W miejsce zmiennych logicznych Dijkstra zaproponował wprowadzenie zmiennych typu całkowitoliczbowego, które mogą przyjmować wartości nieujemne. Do wykonywania operacji na semaforach wprowadzono dwie podstawowe

operacje, oznaczane tradycyjnie przez *P(S)* oraz *V(S)*, gdzie przez *S* oznaczono semafor:

- *V(S)*: inkrementacja zmiennej *S*. Pobranie wartości zmiennej *S* z pamięci, inkrementacja oraz zapis do pamięci jest operacją niepodzielną. Podczas wykonywania tej operacji inne wątki nie mają dostępu do zmiennej *S*,
- *P(S)*: jeśli *S>0,* to dekrementacja zmiennej *S*. Jeśli *S=0* to wątek wykonujący tę operację zostaje wstrzymany do momentu aż dekrementacja będzie możliwa (inny wątek wykona operację *V(S)*). Operacja sprawdzenia wartości zmiennej *S* oraz dekrementacji jest niepodzielna.



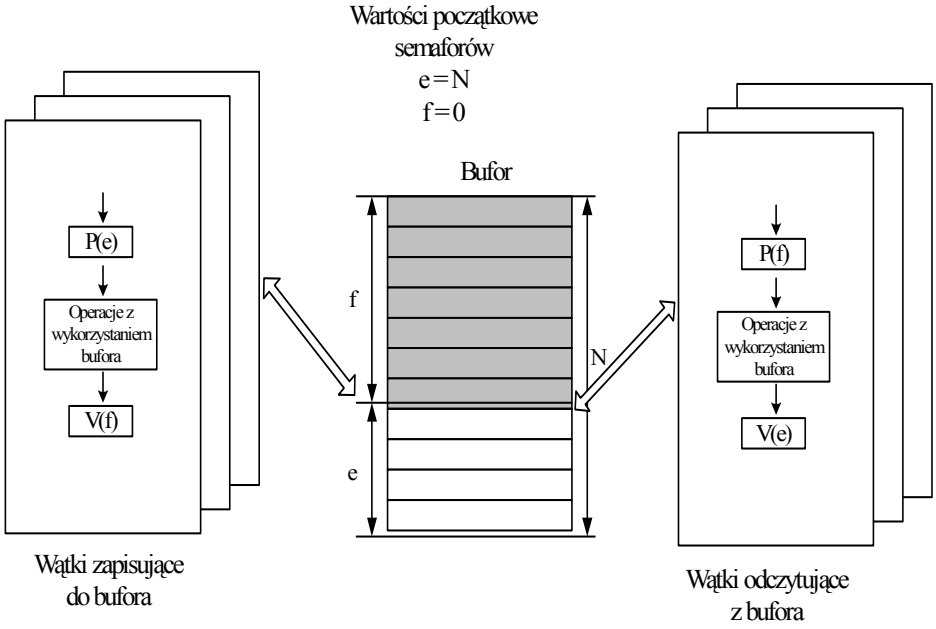
Rys. 4.25. Wzajemne wykluczanie z zastosowaniem funkcji systemowych

W szczególności, gdy semafor *S* przyjmuje tylko wartości 0 i 1, to pełni rolę zmiennej blokującej i jest określany mianem *mutex* (ang. Mutual Exclusion) lub mianem semafora binarnego.

Rozpatrzymy wykorzystanie semaforów na klasycznym przykładzie współdziałania dwóch współbieżnie wykonywanych wątków, z których jeden zapisuje dane do bufora, a drugi odczytuje te dane. Załóżmy, że bufor zawiera N elementów, z których każdy przechowuje jeden rekord. W ogólnym przypadku wątek zapisujący dane do bufora oraz wątek odczytujący mogą

wykonywać swoje operacje z różną prędkością i zwracać się do bufora z różną intensywnością. W jednym okresie intensywność zapisu może być większa od szybkości odczytu danych z bufora, podczas gdy w innym może być odwrotnie. Aby współpraca obydwu wątków odbywała się prawidłowo, wątek wpisujący dane do bufora powinien zostać zatrzymany gdy bufor zostanie zapełniony, zaś wątek odczytujący dane z bufora należy zatrzymać w przypadku gdy bufor jest pusty.

W celu synchronizacji wątków wprowadzimy dwa semafory: *e* – o wartości równej liczbie wolnych elementów bufora (początkowa wartość *e=N*) oraz *f* – o wartości określającej liczbę zajętych elementów bufora (wartość początkowa *f=0*). Współpraca wątków może być przedstawiona następująco (rys. 4.26).



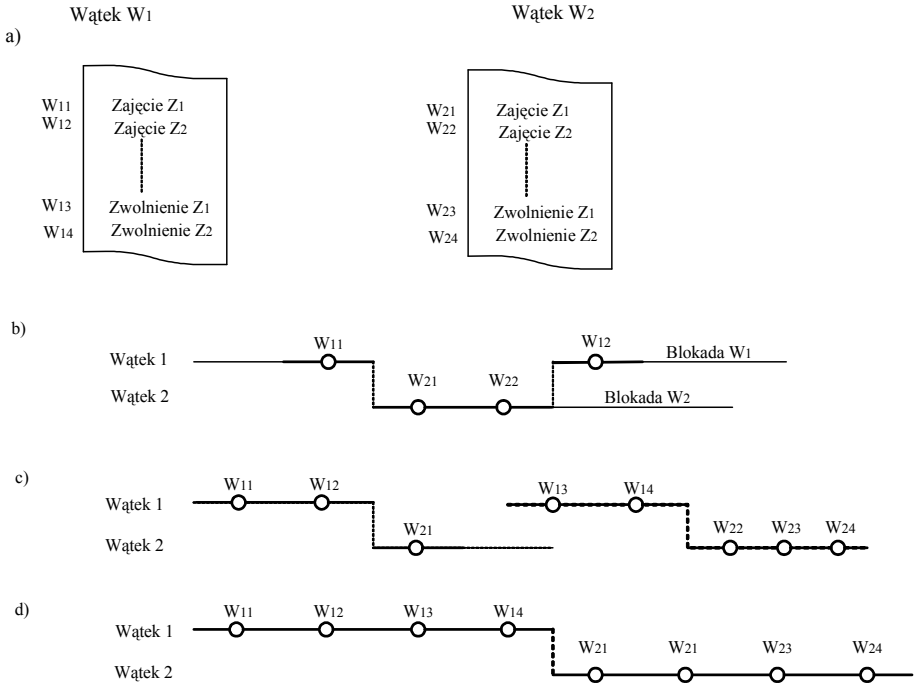
Rys. 4.26. Zastosowanie semaforów do synchronizacji wątków

Wątek wpisujący dane do bufora w pierwszej kolejności wykonuje operację *P(e)*, sprawdzając w ten sposób, czy w buforze znajdują się wolne elementy. Jeśli semafor *e* ma wartość zero (brak wolnych elementów w buforze), to wątek wpisujący dane do bufora zostaje zablokowany. W przeciwnym wypadku wartość semafora *e* zostaje zmniejszona o 1, do kolejnego elementu bufora zostają zapisane dane, a następnie zwiększona zostaje liczba zajętych elementów bufora (operacja *V(f)*). Wątek odczytujący dane z bufora wykonuje operacje analogiczne z tą różnicą, że zaczyna pracę od sprawdzenia liczby elementów zapełnionych (operacja *P(f)*), a po odczytaniu danych zwiększa liczbę wolnych elementów bufora (operacja *V(e)*). Znacznie korzystniejsze jest wykorzystanie semaforów niż zmiennych blokujących, gdyż pozwala na synchronizację większej liczby wątków wpisujących oraz odczytujących dane (łączna liczba wątków wynosi N).

4.4.5. Blokady (ang. deadlocks)

Blokady wątków mogą wystąpić w przypadku, gdy dwa wątki wykorzystują dwa te same zasoby. Na rys. 4.27a przedstawiono fragmenty programów realizowanych w obydwu wątkach. Wątek W_1 zwraca się najpierw do zasobu Z_1 a następnie do zasobu Z_2, podczas gdy wątek W_2 odwrotnie – najpierw do zasobu Z_2 a następnie do zasobu Z_1. Załóżmy, że w pierwszej kolejności do zasobu Z_1 zgłosił się wątek W_1 i ustawił związaną z tym zasobem zmienną blokującą. Po tej operacji wątek W_1 został wywłaszczony, a następnie system operacyjny przekazał procesor wątkowi W_2, który zgłasza się do zasobu Z_2 i ustawia zmienną blokującą związaną z tym zasobem. W następnej kolejności wątek W_2 zgłasza się do zasobu Z_1, a ponieważ zasób ten jest zajęty przez wątek W_1, zostaje przez system operacyjny zatrzymany. Załóżmy, że system operacyjny przydziela następnie procesor wątkowi W_1, który zgłasza się do zasobu Z_2, a ponieważ ten jest zajęty przez wątek W_2, zostaje przez system operacyjny zatrzymany. W ten sposób obydwa wątki zostają zatrzymane przez system operacyjny i mogą w tym stanie przebywać dowolnie długo (rys. 4.27b). W celu uwolnienia blokady wątków jest konieczna ingerencja systemu operacyjnego.

W zależności od względnej szybkości wykonywania wątków W_1 i W_2 mogą mieć miejsce różne sytuacje, takie jak blokada (rys. 4.27b), tworzenie kolejek do poszczególnych zasobów (rys. 4.27c), niezależne wykorzystywanie obydwu zasobów (rys. 4.27d).



Rys. 4.27. Wzajemna blokada wątków

W przedstawionym przykładzie blokada występowała dla dwóch wątków, jednak opisana sytuacja może mieć także miejsce gdy wiele wątków wykorzystuje szereg wspólnych zasobów. Problemami identyfikacji blokad oraz ich rozwiązywania zajmuje się system operacyjny.

4.4.6. Synchronizacja wątków w systemie Windows NT

Win32 API udostępnia cztery mechanizmy synchronizacji wątków: *sekcje krytyczne (*ang. *critical sections), wykluczenia wzajemne,* zwane także *muteksami (*ang. *mutex*), *semafory (ang. semaphores)* oraz *zdarzenia (*ang. *events).*

1. Sekcje krytyczne

Sekcja krytyczna w Win32 jest zmienną typu rekordowego TRTLCriticalSection, zadeklarowaną w programie jako zmienna globalna procesu, dostępną dla wszystkich wątków tego procesu. Na skutek tego sekcje krytyczne w Windows NT mogą być stosowane wyłącznie do synchronizacji wątków wchodzących w skład jednego procesu. Microsoft ukrywa strukturę rekordu TRTLCriticalSection ponieważ zmienia się ona w zależności od platformy sprzętowej. Dla platformy sprzętowej opartej na procesorze Pentium sekcja krytyczna zawiera licznik oraz pole przechowujące uchwyt przebywającego w niej wątku.

Z zastosowaniem sekcji krytycznych można wykonywać następujące operacje:

- Inicjacja sekcji krytycznej, wykonywana przez następującą procedurę API:

```
procedure InitializeCriticalSection(var
lpCriticalSection:RTLCriticalSection);
```

- Krytyczny fragment kodu programu musi być poprzedzony operacją wejścia do sekcji krytycznej, realizowaną przez procedurę:

```
procedure EnterCriticalSection(var
lpCriticalSection:RTLCriticalSection);
```

W danej chwili wewnątrz określonej sekcji krytycznej może przebywać co najwyżej jeden wątek – pozostałe wątki zamierzające do niej wejść (wywołujące w stosunku do niej procedurę `EnterCriticalSection`) zostaną przez system operacyjny wstrzymane.

- Krytyczny fragment kodu musi kończyć się operacją wyjścia z sekcji krytycznej, realizowaną przez procedurę:

```
procedure LeaveCriticalSection(var
lpCriticalSection:TRTLCriticalSection);
```

Operacja ta umożliwia wejście do sekcji krytycznej któremuś z oczekujących wątków.

- Gdy sekcja krytyczna nie jest już potrzebna, należy ją zwolnić przy pomocy następującej procedury:

```
procedure DeleteCriticalSection(var
      lpCriticalSection:TRTLCriticalSection);
```

2. Muteksy (wzajemne wykluczenia)

Mechanizm wzajemnego wykluczania (ang. mutual exclusion) jest analogiczny do sekcji krytycznej, jednak różni się dwoma istotnymi właściwościami:

- mutex jest globalnym obiektem Win32 API, reprezentowanym przez uchwyt (ang. handle), co powoduje, że przy jego pomocy można synchronizować wątki odrębnych procesów (w przeciwieństwie do sekcji krytycznej, której mogą używać wyłącznie wątki tego samego procesu).
- mutex jest dostępny dla wszystkich wątków wszystkich procesów poprzez swą nazwę symboliczną.

Utworzenie muteksu odbywa się przy pomocy następującej funkcji:

```
Function CreateMutex(
   lpMutexAttributes:PSecurityAttributes;
   bInitialOwner: BOOL; lpName:Pchar): Thandle;
```

Parametr `lpMutexAttributes` jest wskaźnikiem do struktury określającej tzw. atrybuty bezpieczeństwa; podanie w wywołaniu funkcji pustego wskaźnika spowoduje przyjęcie atrybutów domyślnych.

Parametr `bInitialOwner` określa właściciela muteksu – wartość TRUE oznacza, że właścicielem muteksu jest wątek, który go utworzył, natomiast wartość FALSE oznacza, że utworzony muteks nie posiada właściciela.

Parametr `lpName` jest globalną nazwą w systemie operacyjnym identyfikującą muteks; wartość NUL powoduje utworzenie muteksu nienazwanego. Funkcja CreateMutex() poszukuje w systemie istniejącego już muteksu o podanej nazwie i w przypadku jego znalezienia tworzy do niego dodatkowy uchwyt, w przeciwnym wypadku tworzy nowy muteks i zwraca do niego uchwyt.

Zwolnienie muteksu polega na wykonaniu procedury:

```
procedure CloseHandle(hHandle: Thandle)
```

zwalniającej uchwyt do muteksu.

Oczekiwanie wątku na dostęp do zasobu udostępnianego przez muteks realizowane jest przy pomocy następującej funkcji API:

```
function WaitForSingleObject(
             hHandle: Thandle;
             dwMiliseconds:DWORD): DWORD;
```

Funkcja powoduje, że wątek czeka tak długo, aż wątek reprezentowany przez uchwyt `hHandle` znajdzie się w tzw. stanie sygnalnym, nie dłużej jednak niż przez czas określony przez parametr `dwMiliseconds` (muteks jest w stanie sygnalnym gdy żaden wątek nie używa zasobu przezeń chronionego). Limit czasu oczekiwania określony przez drugi parametr (w milisekundach) określa maksymalny czas oczekiwania na zasób. Podanie wartości INFINITE oznacza oczekiwanie do skutku.

Funkcja `WaitForSingleObject` wraca jedną z trzech wartości:

`WAIT_ABANDONED` – wątek będący właścicielem muteksu zakończył się, nie dokonując jego zwolnienia (muteks porzucony). Właścicielem muteksu staje

się wątek macierzysty wątku, który zakończył działanie. W wyniku działania funkcji `WaitForSingleObject` muteks nie znajduje się w stanie sygnalnym.

`WAIT_OBJECT_0` – muteks znajduje się w stanie sygnalnym (wykonywany wątek uzyskuje dostęp do zasobu chronionego, uniemożliwiając dostęp innym wątkom).

`WAIT_TIMEOUT` – został wyczerpany limit czasu oczekiwania, muteks nie znajduje się w stanie sygnalnym.

Po doczekaniu się wątek staje się właścicielem muteksu i może być wykonywany krytyczny fragment kodu, który należy zakończyć procedurą `ReleaseMutex(hHandle:Thandle)`.

Jeżeli muteks nie ma przydzielonego właściciela ani nazwy, to jest identyfikowany jedynie przez uchwyt. Jego właścicielem staje się wątek, który wykonuje w stosunku do niego funkcję `WaitForSingleObject()`, używając uchwytu do muteksu. Stosunek własności ustaje w momencie wykonania przez wątek funkcji `ReleaseMutex()`. Oznacza to, że muteksy nienazwane mogą być używane jedynie przez wątki należące do tego samego procesu.

3. Semafory

Pod względem funkcjonalności semafor przypomina muteks, z tą różnicą, że posiada dodatkowe wyposażenie w postaci licznika. Niezerowa, dodatnia wartość tego licznika oznacza stan sygnalny semafora. Do utworzenia semafora służy następująca funkcja:

```
function CreateSemaphore
      (lpSemaphoreAttributes: PsecurityAttributes;
       lInitialCount, lMaximumCount: Longin;
       lpName: Pchar): Thandle;
```

Parametry `lpSemaphoreAttributes` i `lpName` mają takie same znaczenie jak w przypadku funkcji CreateMutex.

Parametr `lInitialCount` określa początkową wartość licznika semafora.

Każdorazowe wystąpienie semafora jako parametru wywołania funkcji `WaitForSingleObject()` zmniejsza o jeden jego licznik, chyba że ma w tym momencie wartość zero. Natomiast każde wywołanie funkcji `ReleaseSemaphore()` zwiększa o jeden stan licznika semafora.

Wartość `lMaximumCount` określa początkowy stan licznika semafora i oznacza maksymalną liczbę zasobów, jakie chroni semafor. Wyjaśnia to typowe zastosowanie semafora – służy on do synchronizacji dostępu do zasobu, który może być współdzielony przez co najwyżej zadaną a priori liczbę procesów.

W przypadku gdy sekcja krytyczna programu może być wykonywana jednocześnie przez jeden wątek, początkowa wartość licznika semafora powinna wynosić jeden.

Analogicznie jak dla muteksów, funkcja synchronizująca wątki z zastosowaniem semafora ma postać:

```
function WaitForSingleObject( hHandle: Thandle;
              dwMiliseconds:DWORD): DWORD;
```

i musi być wykonana na początku sekcji krytycznej.

Wątek opuszczając sekcję krytyczną sygnalizuje ten fakt poprzez wykonanie funkcji:

```
function ReleaseSemaphore(hSemaphore:Thandle;
                  lReleaseCount:Longin;
                  lpPreviousCount:Pointer):BOOL;
```

Pierwszym parametrem tego wywołania jest oczywiście uchwyt semafora. Parametr `lReleaseCount` określa o ile należy zwiększyć wartość licznika semafora – musi to być wartość dodatnia, niekoniecznie równa jeden. Ostatni parametr wywołania funkcji `ReleaseSemaphore` – `lpPreviousCount` – umożliwia wskazanie zmiennej typu longint, do której wpisana zostanie poprzednia wartość licznika semafora (sprzed wywołania funkcji) – w Win32 nie istnieje możliwość odczytania bieżącego stanu licznika semafora i można to wykonać jedynie po fakcie jego uwolnienia.

Rozdział V
ZARZĄDZANIE PAMIĘCIĄ OPERACYJNĄ

5.1. Funkcje systemu operacyjnego związane z zarządzaniem pamięcią

Pamięć operacyjna jest podstawowym zasobem systemu komputerowego gdyż procesor może realizować programy tylko w przypadku gdy są one umieszczone w pamięci operacyjnej. Operacje związane z zarządzaniem pamięcią mają szczególne znaczenie dla wielozadaniowych systemów operacyjnych, w których pamięci operacyjnej przechowywanych jest jednocześnie wiele programów oraz danych. Funkcje systemu operacyjnego związane z zarządzaniem pamięcią w systemach wielozadaniowych można zgrupować następująco:

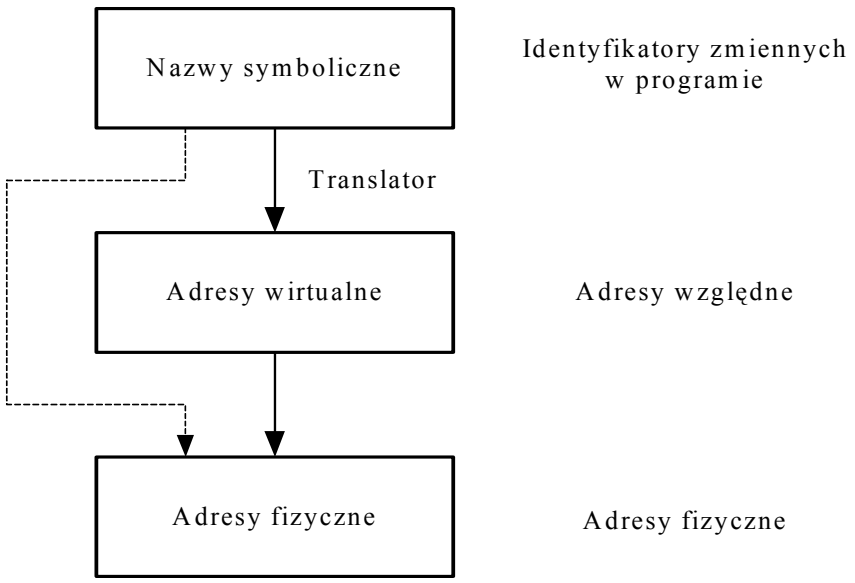
- śledzenie wolnych oraz zajętych obszarów pamięci,
- przydzielanie pamięci tworzonym procesom oraz zwalnianie pamięci po zakończeniu procesu,
- przenoszenie programów oraz danych między pamięcią operacyjną a pamięcią masową,
- przetwarzanie adresów logicznych na adresy fizyczne,
- ochrona pamięci.

5.2. Rodzaje adresów

W celu identyfikacji zmiennych oraz instrukcji, na różnych etapach cyklu życia programu są wykorzystywane różne rodzaje adresów: adresy symboliczne (nazwy zmiennych lub etykiet), adresy wirtualne (logiczne) oraz adresy fizyczne (rys. 5.1):

- *nazwy symboliczne* są nadawane przez programistę podczas tworzenia programu w języku wysokiego poziomu lub w assemblerze,
- *adresy wirtualne* (logiczne) są wyznaczane przez translator podczas tłumaczenia programu z postaci tekstu źródłowego do programu wykonywalnego, zawierającego rozkazy procesora. Ponieważ na etapie tłumaczenia programu nie jest znane miejsce pamięci, w którym będzie umieszczony program podczas jego wykonywania, translator przypisuje poszczególnym nazwom symbolicznym adresy względne, liczone względem początku programu,
- *adresy fizyczne* określają numery komórek pamięci, w których będzie umieszczony program oraz dane podczas realizacji programu.

Ogół adresów wirtualnych procesu określany jest mianem przestrzeni adresów wirtualnych procesu. Każdy proces posiada swoją własną przestrzeń adresów wirtualnych. Przypisanie adresom wirtualnym poszczególnych procesów odpowiadających im adresów fizycznych wykonuje system operacyjny na etapie ładowania programu do pamięci operacyjnej lub etapie wykonywania programu.



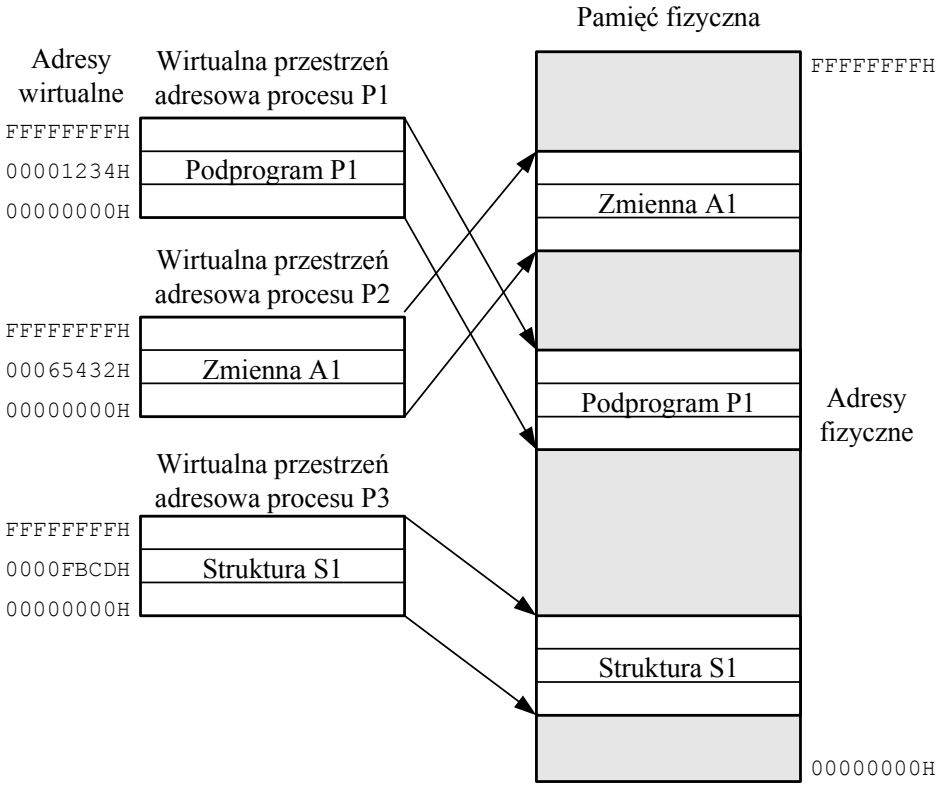
Rys. 5.1. Rodzaje adresów

Na rys. 5.2 przedstawiono wirtualne przestrzenie adresowe poszczególnych procesów oraz odwzorowanie tych przestrzeni na pamięć fizyczną systemu.

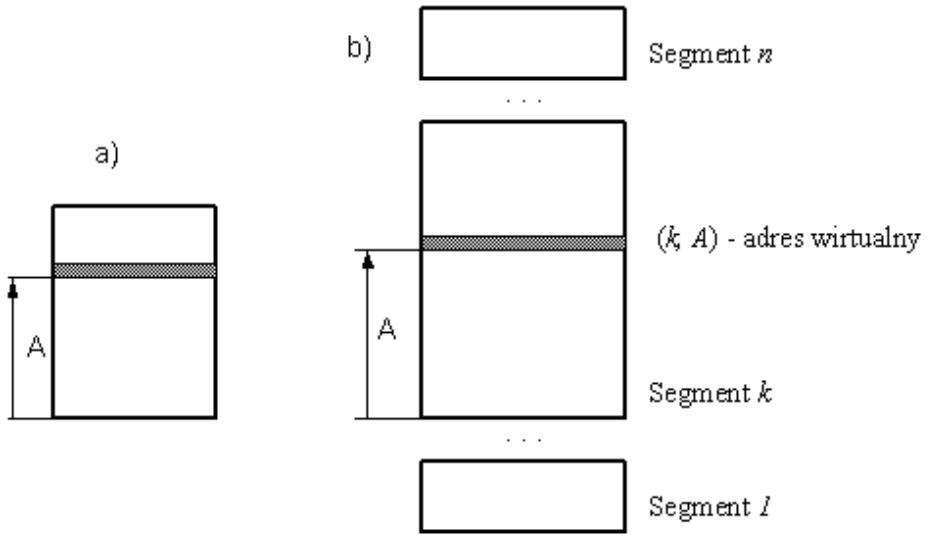
W systemach operacyjnych są stosowane różne struktury przestrzeni adresów wirtualnych. Po pierwsze wirtualna przestrzeń adresowa procesu może być zbliżona do pamięci fizycznej. W takim przypadku wirtualna przestrzeń adresowa jest reprezentowana w postaci liniowego ciągu kolejnych adresów wirtualnych. Taką strukturę przestrzeni adresowej określa się mianem płaskiej. Adres wirtualny w takim przypadku jest przedstawiany przez jedną liczbę, reprezentującą adres względny (przesunięcie – ang. offset) w stosunku do początku wirtualnej przestrzeni adresowej. Adres taki określany jest mianem wirtualnego adresu liniowego.

W przeciwieństwie do płaskiej struktury przestrzeni adresowej, w wielu systemach operacyjnych wirtualna przestrzeń adresowa procesu zostaje podzielona na szereg części, określanych mianem segmentów. Z zasady segmentami są logiczne części procesu: kod (program), dane oraz stos. W takiej sytuacji adres wirtualny jest reprezentowany przez parę liczb (k, A), gdzie k reprezentuje segment, a A – adres względny w segmencie (ang. offset) (rys. 5.3).

Odwzorowanie wirtualnych przestrzeni adresowych poszczególnych procesów, jednocześnie umieszczonych w pamięci, na pamięć fizyczną systemu komputerowego jest zadaniem systemu operacyjnego. Istnieją dwa zasadniczo różniące się podejścia do przekształcania adresów wirtualnych w adresy fizyczne.



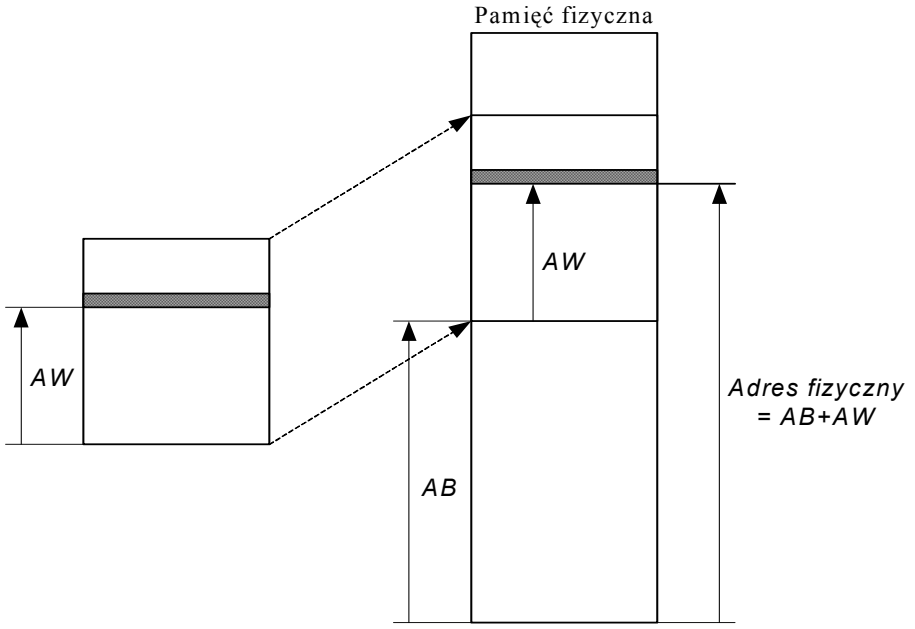
Rys. 5.2. Przestrzenie adresów wirtualnych procesów i odpowiadające im obszary w pamięci fizycznej



Rys. 5.3. Struktury wirtualnych przestrzeni adresowych: płaska (a), segmentowana (b)

W przypadku *statycznego wiązania adresów* operacja zamiany adresów wirtualnych na fizyczne odbywa się jeden raz dla każdego procesu, podczas ładowania programu do pamięci operacyjnej. Operację tę wykonuje program ładujący, wchodzący w skład systemu operacyjnego. Program ten na podstawie informacji o adresie początkowym pamięci fizycznej odczytuje program z pliku dyskowego i umieszcza go (poczynając od podanego adresu) w pamięci, a następnie poddaje analizie i zamienia wszystkie adresy wirtualne w nim występujące (adresy danych oraz rozkazów) na adresy fizyczne.

W przypadku *dynamicznego wiązania adresów* pod określony adres pamięci zostaje załadowany program w takiej postaci, w jakiej został przygotowany przez translator (program zawiera adresy wirtualne) i nie są w nim dokonywane żadne zmiany. Przekształcanie adresów wirtualnych na fizyczne odbywa się następnie podczas wykonywania programu. Aby nie stracić na szybkości, operacje przekształcania adresów muszą być w takim wypadku wspomagane przez procesor. W przypadku płaskiej przestrzeni adresowej zamiana adresu wirtualnego na fizyczny polega na dodaniu do adresu bazowego *AB* (określonego przez system operacyjny) adresu wirtualnego *AW* reprezentowanego w programie (rys. 5.4). Np. podczas realizacji rozkazu `MOV AW,R0` należy wyznaczyć adres fizyczny zmiennej *AW* poprzez wykonanie operacji *AB*+*AW*. W tym celu procesor musi być wyposażony w rejestr, w którym będzie przechowywany aktualny adres bazowy, a także jednostkę arytmetyczną sumującą adresy.



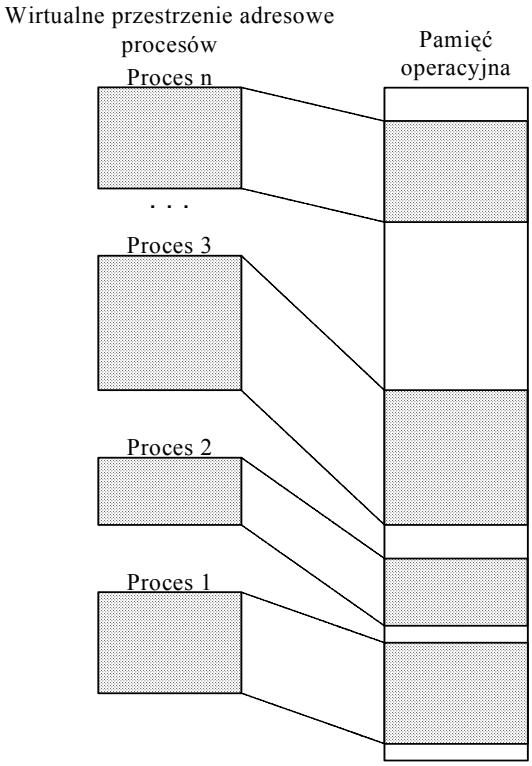
Rys. 5.4. Zasada dynamicznego wiązania adresów

Dynamiczne wiązanie adresów jest rozwiązaniem bardziej elastycznym, gdyż pozwala na przemieszczanie procesów w pamięci podczas ich realizacji (wystarczy skopiować określony obszar pamięci oraz zmienić adres bazowy i wpisać do odpowiedniego rejestru procesora), na co nie pozwala statyczne wiązanie adresów. Jednak statyczne wiązanie adresów jest rozwiązaniem tańszym i nie wymaga wspomagania przez procesor.

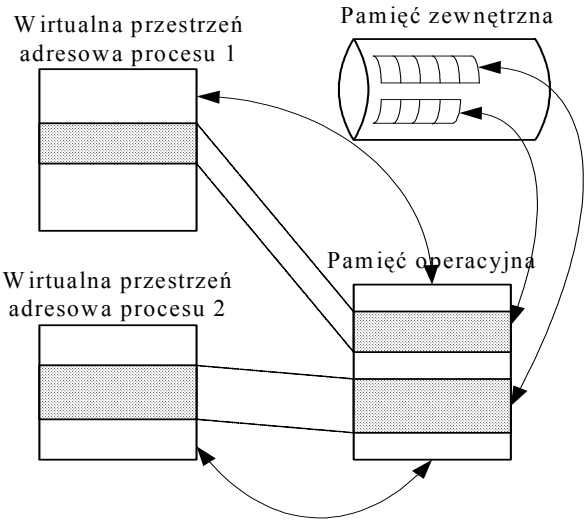
Wirtualna przestrzeń adresowa procesu określa zakres dopuszczalnych adresów wirtualnych, niezbędnych do prawidłowej jego realizacji. Przestrzeń ta jest określona przez translator podczas tworzenia programu wykonywalnego oraz analizy zadeklarowanych danych. Podczas wykonywania programu system operacyjny (wspomagany przez procesor) kontroluje, czy adresy wirtualne nie przekraczają przydzielonej wirtualnej przestrzeni adresowej procesu. W przeciwnym wypadku procesor generuje wyjątek, a system operacyjny wstrzymuje wykonywanie programu. W niektórych systemach operacyjnych proces może zwrócić się do systemu operacyjnego o poszerzenie przydzielonej mu wirtualnej przestrzeni adresowej.

W ogólnym przypadku maksymalny rozmiar wirtualnej przestrzeni adresowej jest określony przez liczbę bitów adresu wirtualnego (np. dla adresu 32-bitowego przestrzeń adresowa wynosi 2^{32} = 4 GB, a dla adresu 48-bitowego – 2^{48} = 256 TB) i nie musi być zgodny z fizyczną przestrzenią adresową danego systemu komputerowego. Może się więc zdarzyć, że fizyczna przestrzeń adresowa może być większa od przestrzeni wirtualnej (rys. 5.5) lub odwrotnie. We współczesnych systemach komputerowych spotykamy się raczej z drugą sytuacją, gdy rzeczywista pojemność pamięci operacyjnej systemu komputerowego jest mniejsza od maksymalnego rozmiaru wirtualnej przestrzeni adresowej. W takim przypadku, w celu przechowywania danych w wirtualnej przestrzeni adresowej, system operacyjny wykorzystuje pamięć masową (rys. 5.6). Jest to zasada funkcjonowania pamięci wirtualnych, stosowanych we wszystkich współczesnych systemach operacyjnych.

Zawartość wirtualnej przestrzeni adresowej procesu (program binarny, przetwarzane dane oraz stos) jest określana mianem *obrazu procesu.* Podczas realizacji procesów, poza wykonywaniem programu użytkowego są także wykonywane programy systemowe, wywoływane jako funkcje systemowe. W celu uproszczenia operacji przekazywania sterowania między programami użytkowymi a programami systemowymi, a także w celu zapewnienia łatwego dostępu modułów systemu operacyjnego do danych użytkowych (np. podczas wyprowadzania danych na urządzenie zewnętrzne), w większości współczesnych systemów operacyjnych ich segmenty są współdzielone z segmentami aktywnego procesu. W ten sposób segmenty systemu operacyjnego oraz segmenty procesu aktywnego tworzą wspólną wirtualną przestrzeń adresową.



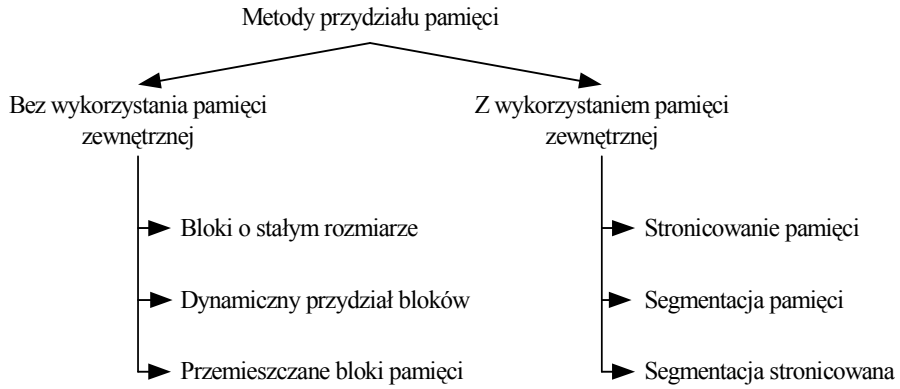
Rys. 5.5. Związek rozmiarów wirtualnej przestrzeni adresowej oraz pamięci fizycznej w przypadku gdy rozmiar wirtualnej przestrzeni adresowej jest mniejszy od pamięci fizycznej



Rys. 5.6. Związek rozmiarów wirtualnej przestrzeni adresowej oraz pamięci fizycznej w przypadku gdy rozmiar wirtualnej przestrzeni adresowej jest większy od pamięci fizycznej

5.3. Algorytmy przydziału pamięci

W rozdziale będą poddane analizie metody przydziału pamięci na różnych etapach rozwoju systemów operacyjnych. Niektóre z nich są szeroko stosowane we współczesnych systemach operacyjnych, podczas gdy inne mają dzisiaj wyłącznie znaczenie historyczne. Na rys. 5.7 przedstawiono ogólną klasyfikację metod przydziału pamięci. Wydzielono dwie grupy metod: metody, w których wykorzystuje się mechanizmy przenoszenia procesów między pamięcią operacyjną a pamięcią zewnętrzną, oraz metody, w których nie wykorzystuje się pamięci zewnętrznej.



Rys. 5.7. Klasyfikacja metod przydziału pamięci

5.3.1. Przydział pamięci oparty o bloki o stałym rozmiarze

Najprostszy sposób zarządzania pamięcią jest oparty na podziale jej na stałe bloki o określonym rozmiarze. Taki podział może być dokonany przez operatora np. podczas startu systemu, a następnie podczas pracy systemu granice między poszczególnymi blokami nie ulegają przemieszczeniom (rys. 5.8). Nowo powołany proces zostaje umieszczony w ogólnej kolejce procesów oczekujących na umieszczenie w pamięci lub w jednej z kolejek związanych z poszczególnymi blokami w pamięci.

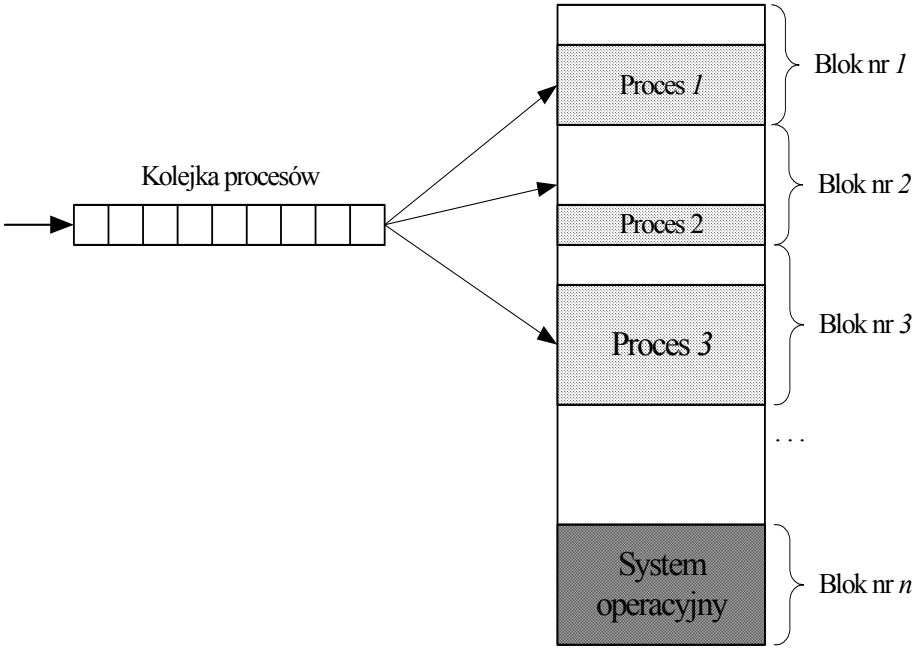
Podsystem zarządzania pamięcią realizuje w tym przypadku następujące operacje:

- porównuje rozmiar pamięci niezbędny dla nowego procesu z rozmiarami wolnych bloków oraz wybiera odpowiedni blok,
- ładuje do zadanego bloku nowy proces oraz ustala adresy.

Blok odpowiedni dla procesu może być w tym przypadku określony już na etapie kompilacji programu, co pozwala na ustalenie adresów fizycznych przez kompilator.

Zaletą tej metody jest prostota realizacji, jednak bardzo istotną wadą jest mała elastyczność. Ponieważ w danym bloku może być wykonywany tylko jeden proces, stopień wieloprocesowości systemu jest ograniczony przez liczbę

bloków w pamięci (proces zajmuje cały blok niezależnie od jego rozmiaru). Dla przykładu, w systemie z ośmioma blokami w pamięci można wykonywać jednocześnie maksymalnie osiem procesów, mimo że procesy mogą zajmować niewielką część pamięci. Z drugiej strony podział pamięci na bloki wyklucza możliwość wykonania procesu o rozmiarze większym od rozmiaru pojedynczego bloku.



Rys. 5.8. Podział pamięci na bloki o stałych rozmiarach

Ten sposób przydziału pamięci był stosowany w pierwszych wielozadaniowych systemach operacyjnych. Współcześnie metodę bloków pamięci o stałym rozmiarze można spotkać w niektórych systemach operacyjnych czasu rzeczywistego, w których jest niezbędna duża szybkość pracy.

5.3.2. Dynamiczny przydział bloków

W tym przypadku każdemu procesowi jest przydzielany niezbędny obszar pamięci na etapie tworzenia procesu. Jeśli w danym momencie w pamięci operacyjnej nie ma wolnego ciągłego obszaru o odpowiednim rozmiarze, to proces nie jest tworzony. Po zakończeniu pracy procesu pamięć po nim zostaje zwolniona, a na to miejsce może zostać załadowany nowy proces. W ten sposób podczas pracy systemu w pamięci występuje szereg obszarów zajętych przez aktualnie realizowane procesy oraz pofragmentowany wolny obszar pamięci.

Na rys. 5.9 przedstawiono stany pamięci w różnych chwilach podczas pracy systemu:

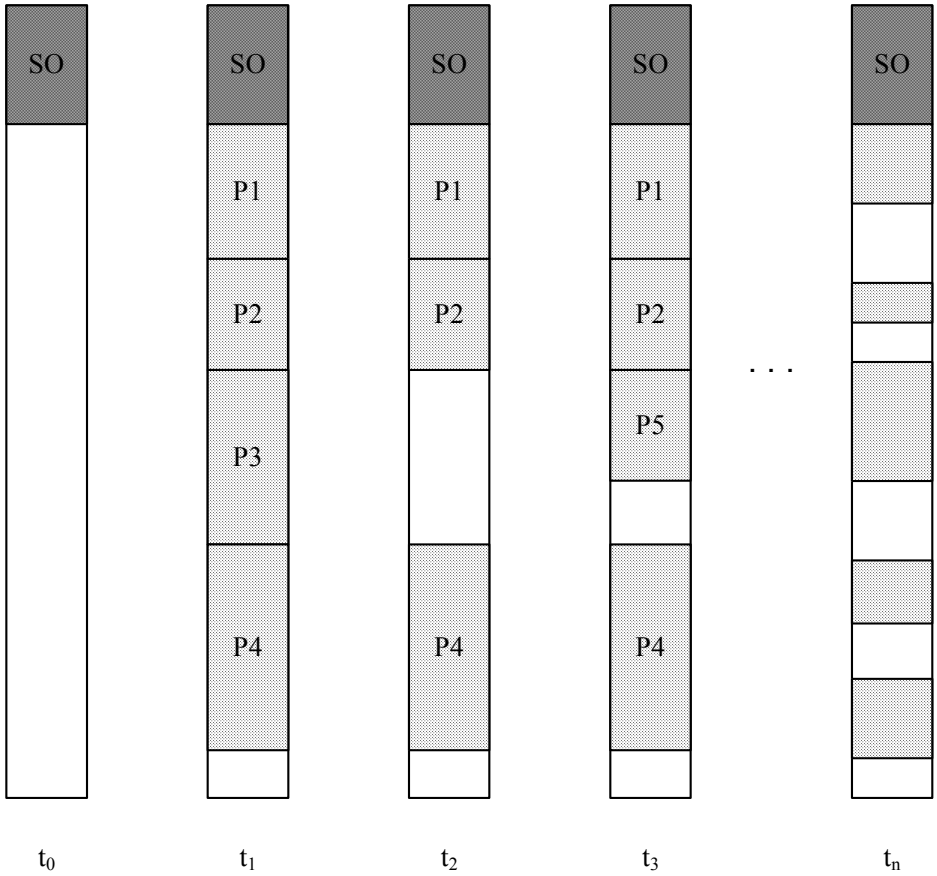
t_0 – w pamięci znajduje się jedynie system operacyjny,

t_1 – do pamięci załadowano cztery procesy,

t_2 – proces P_3 kończy działanie,

t_3 – na miejsce procesu P_3 zostaje załadowany proces P_5,

t_n – stan pamięci po wielu operacjach załadowania procesu do pamięci oraz zakończeniu działania.



Rys. 5.9. Dynamiczny przydział obszarów pamięci

Zadaniem systemu operacyjnego dynamicznie przydzielającego obszary pamięci jest:

- stworzenie i obsługa odpowiednich struktur danych, przechowujących informacje o adresach początkowych i rozmiarach poszczególnych zajętych oraz wolnych obszarów pamięci,

- analiza zapotrzebowania na pamięć przez nowe procesy, przeszukiwanie tablicy wolnych obszarów pamięci oraz wybór obszaru odpowiedniego dla nowego procesu,
- załadowanie programu do określonego obszaru pamięci oraz modyfikacja tablic wolnych i zajętych obszarów pamięci (podczas realizacji proces nie jest przemieszczany w pamięci),
- modyfikacja tablic wolnych oraz zajętych obszarów pamięci po zakończeniu procesu.

W porównaniu z metodą przydziału stałych obszarów pamięci, metoda dynamicznego przydziału pamięci jest znacznie bardziej elastyczna. Charakteryzuje się jednak bardzo istotnym mankamentem, a mianowicie fragmentacją pamięci, która powoduje, że w pamięci powstaje cały szereg niewielkich wolnych obszarów, których nie można wykorzystać. Metoda dynamicznego przydziału pamięci była bardzo szeroko stosowana w latach 60. i 70. poprzedniego stulecia, np. w popularnym systemie OS/360.

5.3.3. Przemieszczane bloki pamięci

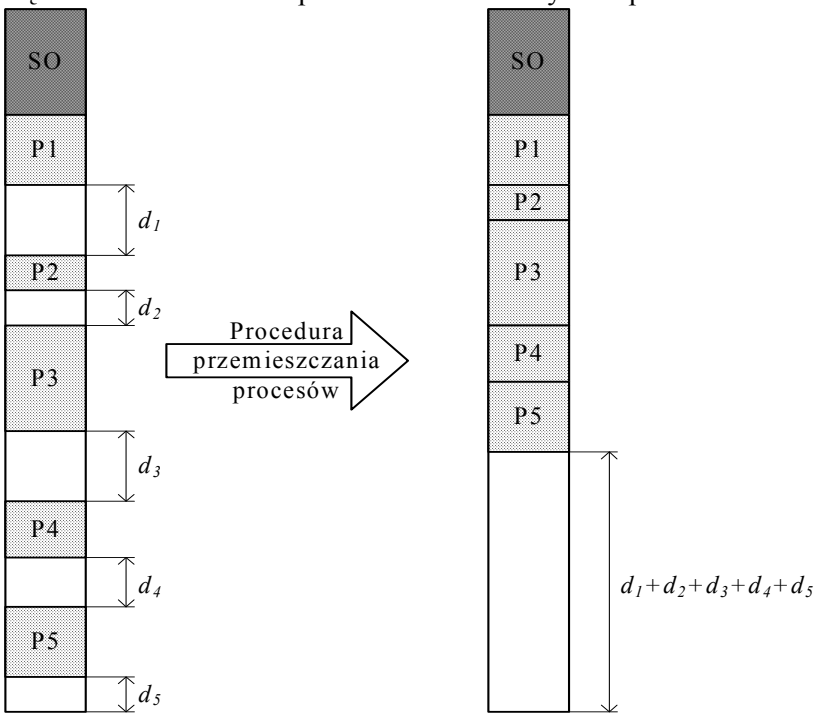
Jedną z metod eliminacji fragmentacji pamięci jest przemieszczanie wszystkich zajętych obszarów pamięci w jednym kierunku (np. w kierunku młodszych adresów) w celu połączenia wszystkich obszarów wolnych (rys. 5.10). W takim przypadku system operacyjny powinien dodatkowo, w stosunku do dynamicznego przydziału pamięci, kopiować zawartość obszarów pamięci zajętych przez poszczególne procesy w inne miejsca oraz modyfikować odpowiednie tablice wolnych i zajętych obszarów pamięci. Operacja ta może być wykonywana po każdym zakończeniu pracy procesu lub tylko w przypadku, gdy dla nowego procesu nie ma miejsca w pamięci. Ponieważ podczas pracy programów następuje przemieszczanie ich w pamięci, więc staje się niezbędne dynamiczne wiązanie adresów, polegające na sprzętowym wyznaczaniu adresów fizycznych.

Procedura przemieszczania obszarów pamięci wymaga znacznego nakładu pracy procesora, co zmniejsza korzyści płynące z łączenia wolnych obszarów pamięci. Metoda ta była stosowana we wczesnych wersjach systemu operacyjnego OS/2.

5.4. Wymiana i pamięć wirtualna

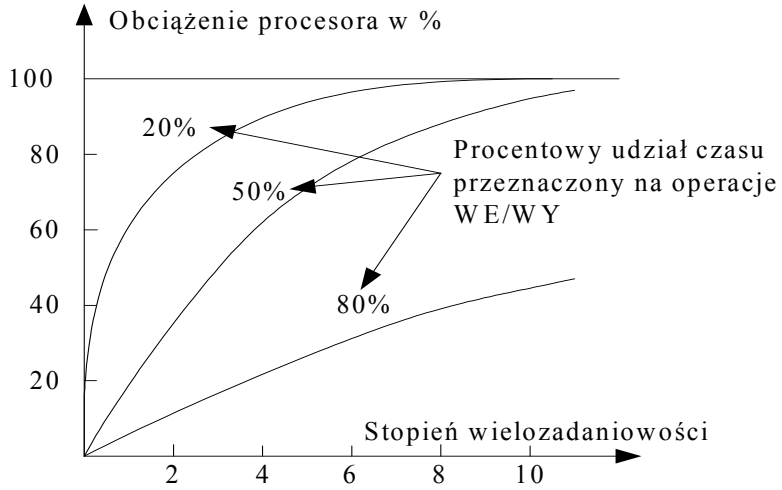
Warunkiem niezbędnym dla realizacji programu jest umieszczenie go w pamięci operacyjnej. Tylko w takim przypadku procesor może pobierać z pamięci oraz realizować poszczególne rozkazy. Rozmiar pamięci operacyjnej systemu komputerowego w istotny sposób wpływa na przebieg procesu obliczeniowego i ogranicza liczbę jednocześnie wykonywanych procesów oraz rozmiary wirtualnych przestrzeni adresowych tych procesów. W przypadku gdy wszystkie procesy realizowane w wielozadaniowym systemie operacyjnym wykonują niewiele operacji WE/WY, już niewielka liczba procesów może spowodować znaczne obciążenie procesora, podczas gdy przy intensywnych

operacjach WE/WY znaczne obciążenie procesora uzyskuje się przy dziesiątkach a nawet setkach procesów realizowanych współbieżnie.



Rys. 5.10. Łączenie wolnych bloków w pamięci poprzez ich przemieszczanie

Na rys. 5.11 przedstawiono przykładową zależność obciążenia procesora od liczby współbieżnie wykonywanych procesów oraz procentowego udziału czasu, podczas którego procesy oczekują w kolejkach do urządzeń WE/WY.



Rys. 5.11. Zależność obciążenia procesora od liczby realizowanych procesów oraz intensywności operacji WE/WY

Znaczna liczba zadań wykonywanych współbieżnie, niezbędnych dla obciążenia procesora, pociąga za sobą konieczność dużej pojemności pamięci operacyjnej systemu komputerowego. W warunkach gdy pojemność pamięci operacyjnej systemu jest niewystarczająca, wprowadza się możliwość takiej organizacji procesu obliczeniowego, w której obrazy wybranych nieaktywnych procesów - w całości lub części – przenosi się czasowo z pamięci operacyjnej do pamięci masowej. Taka podmiana pamięci operacyjnej przez pamięć masową (wirtualizacja) pozwala zwiększyć stopień wieloprogramowości systemu komputerowego, a pojemność pamięci operacyjnej komputera nie ogranicza liczby współbieżnie wykonywanych procesów, gdyż sumaryczna pojemność pamięci niezbędna do przechowania obrazów wszystkich procesów może być znacznie większa od pojemności pamięci operacyjnej.

Dany zasób komputera określa się mianem wirtualnego, gdy dla użytkownika lub programu użytkowego charakteryzuje się właściwościami, których w rzeczywistości nie posiada. W danym przypadku dla użytkownika systemu komputerowego jest dostępna pamięć operacyjna o rozmiarze znacznie przewyższającym rzeczywisty rozmiar pamięci fizycznej. Dzięki takiej właściwości programista może tworzyć programy wykorzystujące pamięć operacyjną o znacznym rozmiarze, a translator – wykorzystując adresy wirtualne – tłumaczy program tak, jakby system komputerowy dysponował pamięcią o takim rozmiarze. W rzeczywistości, podczas wykonywania programu wszystkie kody oraz dane przechowywane są w pamięci masowej i tylko w chwilach, gdy są niezbędne, zostają przeniesione do pamięci operacyjnej. Jest oczywiste, że praca takiej „pamięci operacyjnej" jest znacznie wolniejsza.

Wirtualizacja pamięci jest realizowana w części programowo (przez odpowiednie moduły systemu operacyjnego), a w części sprzętowo przez procesor i jest związana z wykonywaniem następujących zadań:

- rozmieszczenie danych i programów w różnych układach pamięciowych, np. część programu w pamięci operacyjnej, a część na dysku,
- określenie, które obrazy procesów lub ich części mają być przeniesione z pamięci operacyjnej na dysk i odwrotnie,
- przemieszczanie danych między pamięcią operacyjną a dyskiem,
- zamiana adresów wirtualnych na fizyczne.

Istotną cechą takiego rozwiązania jest fakt, że wszystkie powyższe operacje są wykonywane automatycznie przez system operacyjny oraz procesor i nie mają wpływu na logikę pracy aplikacji.

Wirtualizacja pamięci może być zrealizowana na podstawie dwóch różnych zasad:

- wymiana (ang. swapping) – obrazy procesów są w całości przenoszone między pamięcią operacyjną a dyskiem,
- pamięć wirtualna (ang. virtual memory) – między pamięcią operacyjną a dyskiem przenoszone są części (segmenty lub strony) obrazów procesów.

Wymiana jest szczególnym przypadkiem pamięci wirtualnej, pozwalającym na przenoszenie całych obrazów procesów między pamięcią operacyjną a dyskiem. Algorytm ten jednak charakteryzuje się dwoma istotnymi wadami: po pierwsze z zasady istnieje potrzeba wymiany jedynie aktualnie potrzebnych fragmentów kodów oraz danych, a nie całego obrazu procesu. Po drugie wymiana nie rozwiązuje problemu, gdy wirtualna przestrzeń adresowa jednego procesu przekracza rozmiar pamięci fizycznej systemu komputerowego. Z tego powodu algorytm wymiany nie jest praktycznie stosowany we współczesnych systemach komputerowych.

Algorytm pamięci wirtualnej jest aktualnie powszechnie stosowany w systemach operacyjnych. Głównym problemem pamięci wirtualnej, wynikającym z wielokrotnych zmian rozmieszczenia obrazów procesów w pamięci fizycznej, jest wyznaczanie adresów fizycznych na podstawie adresów wirtualnych. We współczesnych systemach operacyjnych są stosowane następujące algorytmy realizacji pamięci wirtualnej:

- segmentacja pamięci polegająca na przydzielaniu miejsca w pamięci dla poszczególnych logicznych części procesu (kod, dane oraz stos), określanych mianem segmentów. Poszczególne segmenty posiadają różne rozmiary oraz zajmują ciągłe obszary w pamięci. Wymiana danych między pamięcią operacyjną a dyskiem odbywa się całymi segmentami,
- stronicowanie pamięci polegające na przydzielaniu miejsca w pamięci porcjami o określonym rozmiarze (strony). Wirtualna przestrzeń adresowa procesu zostaje podzielona na strony, które następnie zostają umieszczone w aktualnie wolnych ramkach pamięci fizycznej. Wymiana danych między pamięcią operacyjną a dyskiem odbywa się stronami o określonym rozmiarze (np. 4 kB),
- segmentacja stronicowana polegająca na dzieleniu obrazu procesu na poszczególne segmenty, które z kolei są dzielone na strony. Jednostką wymiany danych między pamięcią operacyjną a dyskiem jest strona.

W przypadku gdy suma wirtualnych przestrzeni adresowych obrazów wszystkich procesów nie przekracza rozmiaru pamięci fizycznej systemu komputerowego, poszczególne segmenty lub strony zostają w całości umieszczone w pamięci operacyjnej i nie istnieje potrzeba ich wymiany między pamięcią operacyjną a pamięcią masową. Pozostaje jednak problem rozmieszczenia w pamięci fizycznej poszczególnych segmentów oraz stron wchodzących w skład wirtualnych przestrzeni adresowych procesów.

Jeśli suma wirtualnych przestrzeni adresowych obrazów wszystkich procesów przekracza rozmiar pamięci fizycznej systemu komputerowego, powstaje konieczność wymiany segmentów oraz stron między pamięcią operacyjną a pamięcią masową. W celu przechowywania segmentów oraz stron poza pamięcią operacyjną system operacyjny tworzy w pamięci dyskowej specjalny obszar wymiany (partycja wymiany) lub plik wymiany. Rozmiar tego obszaru (pliku) wymiany może określać administrator systemu.

5.4.1. Segmentacja pamięci

W wyniku segmentacji pamięci wirtualna przestrzeń adresowa procesu zostaje podzielona na części logiczne (kod programu, dane oraz stos) o rozmiarach zależnych od potrzeby. Poszczególne części logiczne są określane mianem segmentów i mogą zawierać: program główny, podprogram, zbiór procedur, zbiór danych, stos programu itp. Podział wirtualnej przestrzeni adresowej na poszczególne segmenty może być wykonany przez programistę, jak to ma miejsce podczas tworzenia programu w assemblerze, lub przez kompilator (w przypadku programowania w językach wysokiego poziomu). Maksymalny rozmiar segmentu jest określony przez długość adresu wirtualnego (liczbę bitów w adresie) określającego offset w segmencie. Np. dla adresu 32-bitowego maksymalny rozmiar segmentu wynosi 2^{32} = 4 GB, a dla adresu 16-bitowego – 2^{16} = 64 kB. W przypadku gdy wirtualna przestrzeń adresowa procesu zawiera N segmentów, każdy z segmentów może posiadać rozmiar równy maksymalnemu rozmiarowi segmentu. Dla każdego z segmentów jest określony adres wirtualny w postaci pary liczb określających numer segmentu oraz adres względny w segmencie (offset).

Podział wirtualnej przestrzeni adresowej procesu na segmenty daje możliwość współdzielenia określonych segmentów przez wiele procesów, zawierających np. wspólne dane, czy podprogram wykonywany w wielu procesach (rys. 5.12).

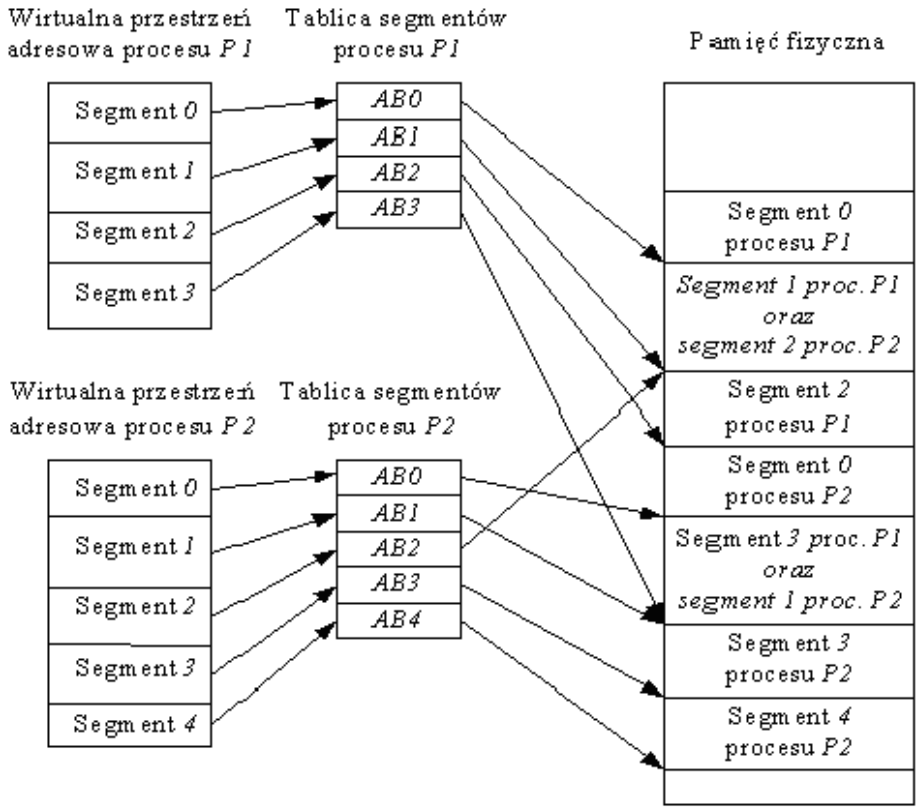
Segmentację pamięci rozpatrzymy dla dwóch przypadków:

1. Suma wirtualnych przestrzeni adresowych wszystkich procesów nie przekracza rozmiaru pamięci fizycznej,
2. Rozmiar pamięci operacyjnej jest niewystarczający dla umieszczenia w niej obrazów wszystkich procesów.

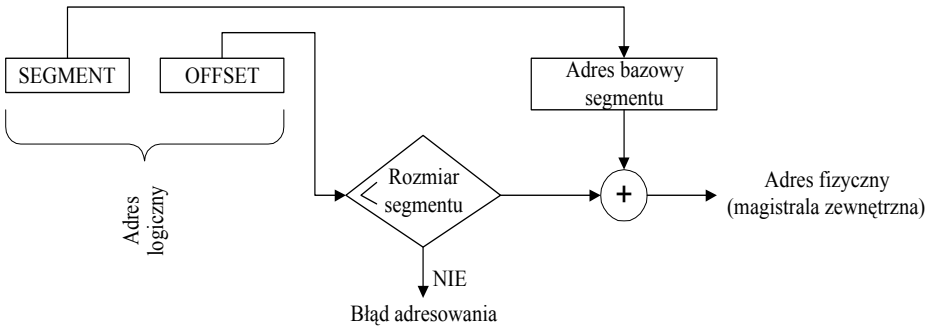
W pierwszym przypadku dla poszczególnych segmentów nowo tworzonego procesu system operacyjny rezerwuje ciągłe obszary pamięci fizycznej o odpowiednich rozmiarach. Dla każdego segmentu system operacyjny tworzy nową pozycję w tablicy segmentów, w której zapisuje następujące dane:

- adres bazowy segmentu w pamięci operacyjnej,
- rozmiar segmentu,
- prawa dostępu do segmentu.

Algorytm wyznaczania adresu fizycznego na podstawie adresu wirtualnego, reprezentowanego przez parę liczb: numer segmentu oraz adres względny w segmencie (offset), przedstawiono na rys. 5.13.



Rys. 5.12. Przydział pamięci fizycznej poszczególnym segmentom
AB – adres bazowy segmentu



Rys. 5.13. Algorytm odwzorowania adresu logicznego w adres fizyczny dla segmentacji pamięci

Znając numer danego segmentu, należy sięgnąć do odpowiedniej pozycji w tablicy segmentów, z której zostaje odczytany adres bazowy oraz rozmiar segmentu. Offset, będący częścią adresu wirtualnego, zostaje porównany

z rozmiarem segmentu i w przypadku, gdy jest większy od rozmiaru segmentu, należy wstrzymać realizację procesu, gdyż próbuje on sięgać do pamięci leżącej poza obszarem przydzielonym procesowi. Adres fizyczny jest sumą adresu bazowego oraz offsetu.

W przypadku gdy rozmiar pamięci operacyjnej jest niewystarczający dla umieszczenia całego obrazu nowo utworzonego procesu, do pamięci operacyjnej jest ładowana jedynie część segmentów niezbędnych do rozpoczęcia realizacji procesu, podczas gdy pozostałe segmenty pozostają na dysku. Gdy w trakcie realizacji procesu ma miejsce odwołanie do segmentu, którego aktualnie nie ma w pamięci, procesor generuje wyjątek. W takim przypadku system operacyjny wstrzymuje realizację procesu oraz przydziela procesor innemu procesowi, a równocześnie wykonuje operację załadowania do pamięci brakującego segmentu.

Szczegóły realizacji segmentacji pamięci rozpatrzymy na przykładzie procesora Pentium. Procesor ten może pracować w dwu trybach pracy, różniących się między innymi funkcjonowaniem pamięci wirtualnych: tryb rzeczywisty (ang. real) oraz tryb chroniony (ang. protected). W obydwu trybach pracy segmentacja pamięci funkcjonuje na odmiennych zasadach.

W trybie *REAL* szesnaście starszych bitów adresu bazowego segmentu znajduje się w jednym z czterech rejestrów segmentowych procesora, natomiast cztery najmłodsze bity są równe zeru. W ten sposób adres bazowy segmentu zawiera dwadzieścia bitów, przy zerowych czterech bitach najmłodszych, co oznacza, że adres bazowy segmentu równy jest iloczynowi zawartości odpowiedniego rejestru segmentowego przez 16. Ze względu na to, że w trybie *REAL* offset w segmencie jest 16-bitowy, więc segment posiada stałą długość równą $2^{16} = 64$ kB.

Procesor *Pentium* w trybie *REAL* zawiera cztery rejestry segmentowe, co daje możliwość jednoczesnego wykorzystywania czterech segmentów dla:

- segmentu programu (rejestr CS),
- segmentu stosu (rejestr SS),
- segmentów danych (rejestry DS, ES).

Jeżeli w rozkazie nie wskazano, którego rejestru segmentowego należy użyć do określenia adresu bazowego segmentu, to przyjmowane są tzw. domyślne rejestry segmentowe, które zestawiono w tabeli 5.1. Domyślnego rejestru segmentowego nie trzeba w programie specjalnie zaznaczać.

Tabela 5.1. Domyślne rejestry segmentowe

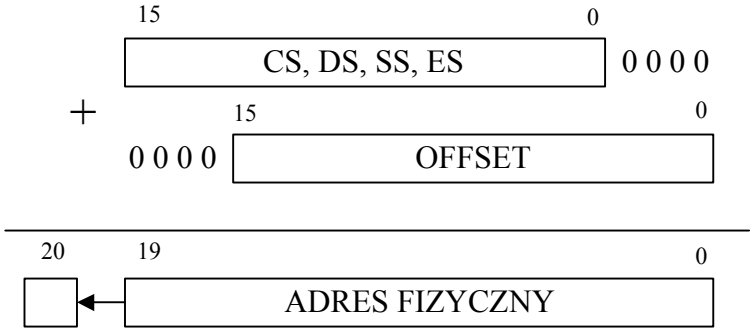
Odwołanie do pamięci	Domniemany rejestr segmentowy	Rejestr zawierający offset
1. Pobieranie rozkazu	CS	IP
2. Operacje na stosie	SS	SP
3. Odczyt/zapis danych	DS	EA*
4. Operacje na łańcuchach (źródło danych)	DS	SI

5. Operacje na łańcuchach (przeznaczenie danych)	ES	DI
6. Adresowanie bazowe rejestrem BP	SS	BP

EA* – adres efektywny, zależny od trybu adresowania

Wyznaczenie adresu fizycznego na podstawie adresu logicznego odbywa się zgodnie ze schematem przedstawionym na rys. 5.14. Np. adresowi logicznemu ABCDh:1234h odpowiada następujący adres fizyczny:

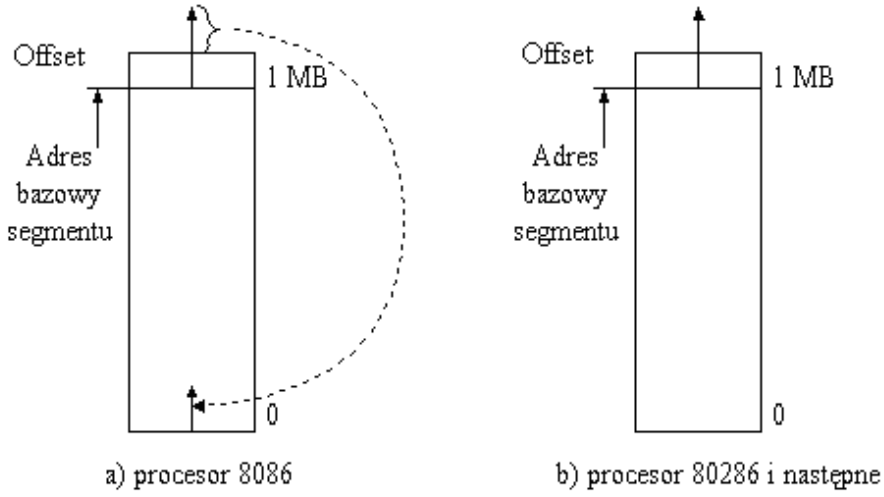
ABCD0h +1234h = ACF04h = 708356_{10}



Rys. 5.14. Wyznaczanie adresu fizycznego na podstawie adresu logicznego realizowane przez procesor Pentium w trybie *REAL*

W przypadku gdy w wyniku wyznaczania adresu fizycznego wystąpi przeniesienie do bitu 20. (bit A20), to w procesorze 8086 jest ono ignorowane (bit 20. adresu nie istnieje), a przestrzeń adresowa „zawija się" (rys. 5.15a). Natomiast w procesorach 80286 i następnych przeniesienie uzyskane przy wyznaczaniu adresu fizycznego zostaje wpisywane do bitu A20. Daje to dodatkową możliwość adresowania w trybie *REAL* obszaru pamięci o pojemności 64 kB, leżącego powyżej obszaru 1 MB (rys. 5.15b).

Ze względu na to, że procesor Pentium w trybie *REAL* dysponuje czterema rejestrami segmentowymi, może jednocześnie obsługiwać cztery segmenty. Segmenty te mogą zajmować oddzielne obszary pamięci fizycznej, mogą się częściowo nakładać, a nawet w pełni pokrywać. Poszczególne segmenty w trybie *REAL* posiadają rozmiar 64 kB ze względu na 16-bitowy offset. Wydaje się to istotnym ograniczeniem, gdyż programy oraz dane mogą być znacznie większe. Nie stanowi to jednak problemu, gdyż w programie można zdefiniować wiele segmentów programów oraz danych, a zawartości rejestrów segmentowych mogą być zmieniane w trakcie wykonywania programu.



Rys. 5.15. Wystąpienie przeniesienia do bitu A20 podczas wyznaczania adresu fizycznego a) w procesorze 8086, b) w procesorach 80286 i następnych

Podczas pracy procesora w trybie chronionym każdy z segmentów programu, danych czy stosu jest opisany zbiorem parametrów określających lokalizację danego segmentu w przestrzeni adresowej pamięci oraz zasady dostępu do tego segmentu. Parametry segmentu są przedstawione w postaci 8-bajtowej struktury danych, określanej mianem deskryptora. Podczas wyznaczania adresu procesor wykorzystuje deskryptor aby określić czy jest dopuszczalne odwołanie do tego segmentu, wyznaczyć adres komórki pamięci (tzw. adres liniowy) oraz sprawdzić czy adres mieści się w zakresie tego segmentu. Deskryptory segmentów przechowywane są w pamięci w postaci tablic deskryptorów. Tablice te mogą mieć rozmiary od 8 bajtów (deskryptor opisujący jeden segment) do 64 kB (8192 deskryptorów).

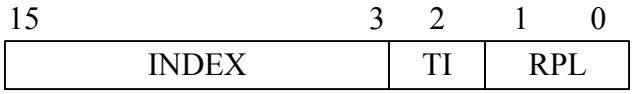
Przewidziano trzy typy tablic deskryptorów:

- GDT (ang. Global Descriptor Table) – globalna tablica deskryptorów,
- LDT (ang. Local Descriptor Table) – lokalna tablica deskryptorów,
- IDT (ang. Interrupt Descriptor Table) – tablica deskryptorów przerwań.

Tablica GDT zawiera deskryptory, które mogą być wykorzystywane przez system operacyjny podczas wykonywania swoich zadań. Tablice LDT, których może być maksymalnie 8192, zawierają deskryptory segmentów wykorzystywanych podczas wykonywania poszczególnych zadań (każde zadanie posiada własną tablicę LDT). Tablica IDT zawiera deskryptory segmentów wykorzystywanych przez procedury obsługi przerwań. W dalszej części skupimy się na opisie adresowania pamięci w trybie chronionym przy pomocy tablic GDT oraz LDT.

Odwołanie do odpowiedniego deskryptora w tablicy deskryptorów jest wykonywane za pomocą selektora, zapisanego w jednym z rejestrów

segmentowych: CS, DS, SS, ES, FS, GS. Selektor jest 16-bitowym rekordem, zawierającym trzy pola (rys. 5.16).



Rys. 5.16. Format selektora

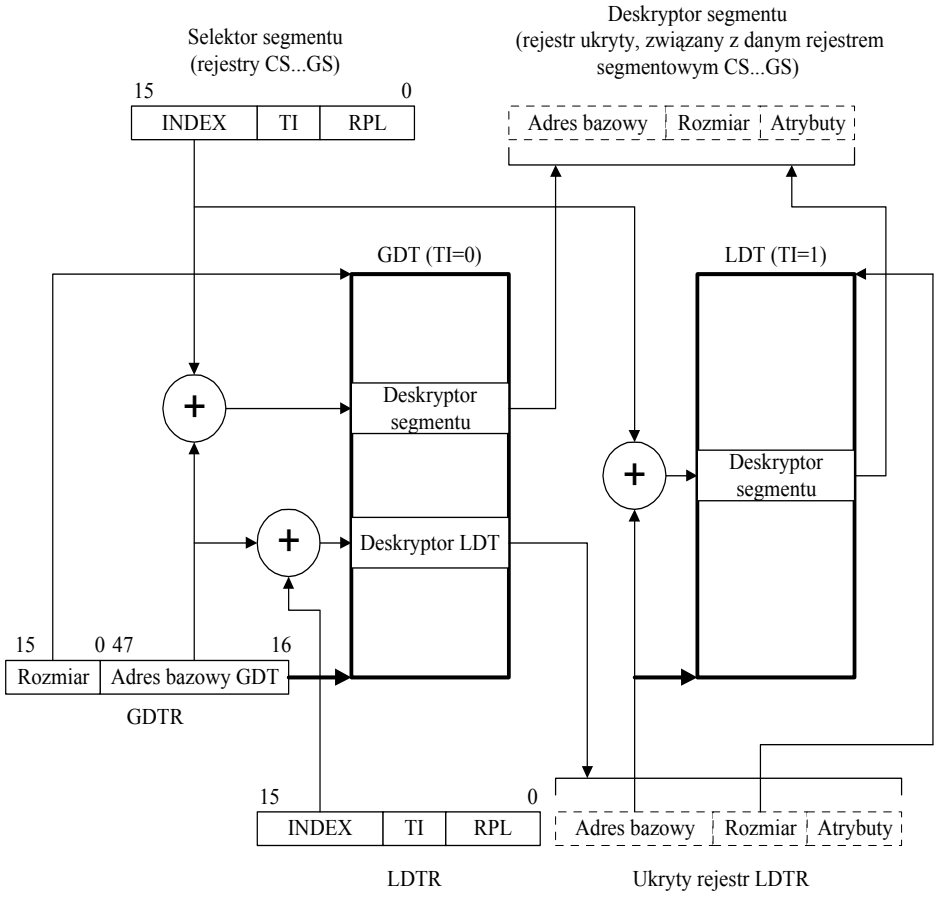
Z każdym z rejestrów segmentowych związany jest 8-bajtowy ukryty rejestr deskryptora, będący kopią odpowiedniego deskryptora z tablicy deskryptorów.

Pole RPL selektora (ang. Requestor's Privilege Level) określa poziom przywileju programu wywołującego. Pole TI selektora (ang. Table Indicator) wskazuje czy selektor dotyczy tablicy globalnej (TI=0), czy lokalnej (TI=1), natomiast 13-bitowe pole INDEX zawiera numer pozycji w tablicy deskryptorów. Do określenia adresu bazowego poszczególnych tablic oraz ich rozmiarów w procesorze przewidziano odpowiednie rejestry: GDTR, IDTR, LDTR. 48-bitowe rejestry GDTR i IDTR zawierają 32-bitowy adres bazowy odpowiedniej tablicy deskryptorów oraz 16-bitowy rozmiar tablicy. Natomiast 16-bitowy rejestr LDTR zawiera selektor, określający numer pozycji w globalnej tablicy deskryptorów, w której deskryptor opisuje umieszczenie w pamięci lokalnej tablicy deskryptorów oraz jej rozmiar. Po załadowaniu do LDTR nowego selektora, z tablicy GDT wybierany jest odpowiedni deskryptor opisujący tablicę LDT, który z kolei jest ładowany do ukrytego rejestru deskryptora. Proces odczytu deskryptora z odpowiedniej tablicy deskryptorów (globalnej GDT lub lokalnej LDT) przedstawiono na rys. 5.17.

Przesłania danych między rejestrami GDTR, LDTR a pamięcią odbywają się przy pomocy instrukcji LGDT, LLDT oraz SGDT, SLDT. Natomiast załadowanie nowego selektora do rejestrów segmentowych DS, ES, FS, GS, SS wykonywane jest instrukcjami LDS, LES, LFS, LGS, LSS. Zmiana selektora w rejestrze segmentowym CS odbywa się wyłącznie podczas wykonywania skoków międzysegmentowych CALL, JMP, a także podczas przełączenia zadania.

Selektor zapisany w jednym z rejestrów segmentowych odnosi się do tablicy GDT (dla TI=0) lub LDT (dla TI=1). Pole INDEX selektora, przesunięte o 3 bity w lewo (pomnożone przez 8), określa adres względny deskryptora w danej tablicy. Uzyskany adres względny jest następnie porównywany z rozmiarem tablicy, przechowywanym w rejestrze GDTR oraz ukrytym rejestrze LDTR. Jeśli adres względny przekracza rozmiar tablicy, to generowany jest wyjątek informujący o naruszeniu systemu ochrony. W przeciwnym wypadku adres względny deskryptora dodawany jest do adresu bazowego tablicy zapisanego w rejestrze GDTR (tablica GDT) lub rejestrze LDTR (tablica LDT) i w ten sposób otrzymywany jest adres pamięci, pod którym zapisany jest deskryptor segmentu.

Adres logiczny w trybie chronionym jest reprezentowany przez selektor zapisany w jednym z rejestrów segmentowych oraz offset będący adresem względnym w segmencie.

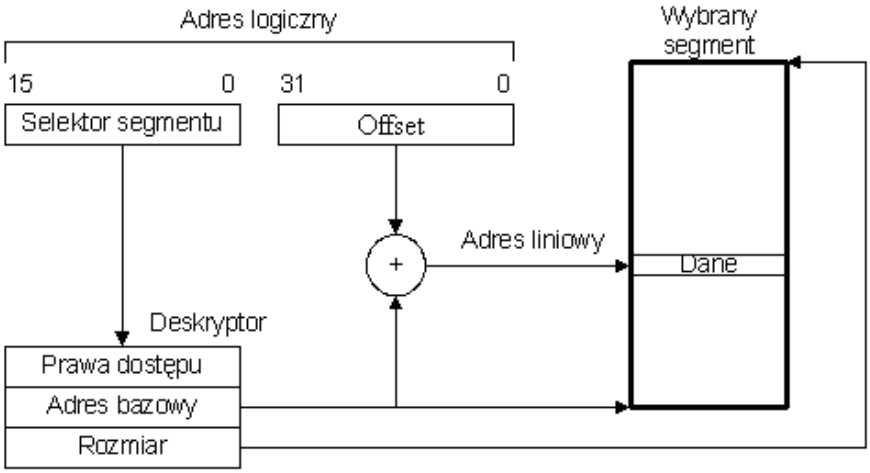


Rys. 5.17. Pobranie deskryptora segmentu z tablicy GDT oraz LDT

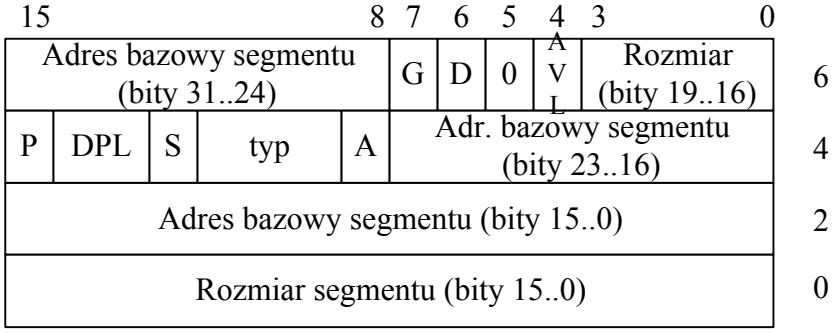
Na podstawie adresu logicznego, w oparciu o deskryptor segmentu pobrany z tablicy deskryptorów, procesor wyznacza adres liniowy. Algorytm wyznaczania adresu liniowego na podstawie adresu logicznego w trybie chronionym przedstawiono na rys. 5.18. W przypadku gdy system operacyjny nie wykorzystuje segmentacji pamięci, adresy bazowe wszystkich segmentów są ustawione na zero, a rozmiary segmentów na 4 GB. Przy wyłączonym układzie stronicowania pamięci adres liniowy jest przekazywany na magistralę zewnętrzną procesora i staje się adresem fizycznym. Jeżeli natomiast jest włączone stronicowanie, adres liniowy poddawany jest dalszemu przetwarzaniu.

Format deskryptora przedstawiono na rys. 5.19. 32-bitowy adres bazowy oraz 20-bitowy rozmiar segmentu zapisane są fragmentami w różnych bajtach

deskryptora. Wynika to z konieczności zapewnienia kompatybilności z procesorem 80286, w którym wprowadzono po raz pierwszy tryb chroniony.



Rys. 5.18. Formowanie adresu liniowego



Rys. 5.19. Format deskryptora segmentu pamięci

Poszczególne bity szóstego bajtu deskryptora opisują atrybuty segmentu i mają następujące znaczenie:

G – bit ziarnistości (ang. granularity) określa w jakich jednostkach podano rozmiar segmentu: dla G=0 w bajtach, dla G=1 w 4 kB. W ten sposób segment może mieć maksymalny rozmiar do 2^{20} = 1 MB dla G=0 oraz do 2^{20} * 4 kB = 4 GB dla G=1.

D – bit określający długość słowa (ang. default); dla D=0 słowo 16-bitowe, D=1 słowo 32-bitowe.

AVL – bit dostępny dla programu (ang. available to software) może być ustawiany przez program użytkowy.

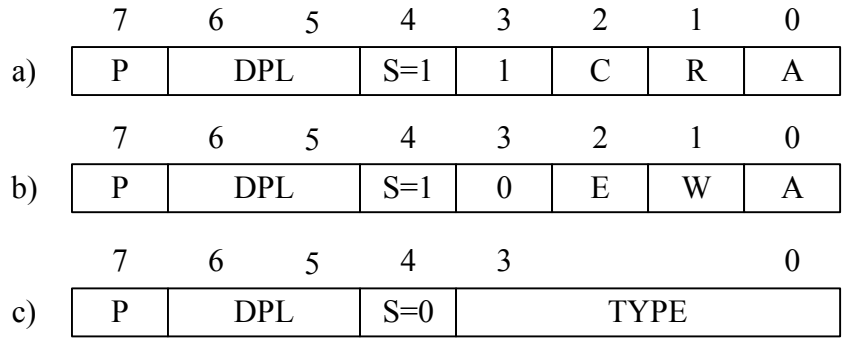
Bajt nr 5 deskryptora określa prawa dostępu do danego segmentu. Format tego bajtu zależy od typu segmentu, jednak pola P, DPL oraz S występują we wszystkich formatach.

P – bit obecności (ang. present) określa czy dany segment jest aktualnie w pamięci. Jeśli P=0 to segment jest nieobecny, a stan pozostałych informacji w deskryptorze może być dowolny. W przypadku, gdy do rejestru deskryptora wpisywany jest deskryptor z zerową wartością bitu P, to generowany jest przez procesor wyjątek braku segmentu.

DPL – 2-bitowe pole określające poziom uprzywilejowania (ochrony) segmentu (ang. Descriptor Privilege Level). W zależności od wzajemnej relacji DPL oraz RPL, zapisanych na dwóch najmłodszych bitach selektora, procesor umożliwia lub zabrania dostępu programowi do danego segmentu.

S – bit określający rodzaj segmentu. S=1 dla deskryptorów segmentów danych oraz segmentów programu. Natomiast deskryptory systemowe, związane z tablicami LDT, segmentami stanu zadania TSS, furtkami czy programami obsługi wyjątków oraz przerwań, posiadają bit S=0.

Formaty bajtu dostępu dla deskryptorów segmentów programu i danych różnią się bitem nr 3: 1 dla segmentów programowych, 0 dla segmentów danych. Pozostałe bity bajtu dostępu dla deskryptorów segmentów programu oraz segmentów danych przedstawiono na rys. 5.20.



Rys. 5.20. Format bajtu dostępu dla: a) deskryptorów programu, b) deskryptorów danych, c) deskryptorów systemowych

A – bit dostępu (ang. accessed), ustawiany na wartość 1 po każdej operacji dostępu do danego segmentu. Jest wykorzystywany przez system operacyjny podczas operacji z pamięcią wirtualną. System operacyjny cyklicznie odczytuje bity A wszystkich wykorzystywanych segmentów, a segmenty dla których przez dłuższy okres czasu bit A=0, zostają przenoszone na dysk.

R – bit określający czy kod zapisany w segmencie programu może być wyłącznie wykonywany (R=0), czy też będzie możliwe jego odczytanie np. w celu skopiowania (R=1). Danych zapisanych w segmencie programu nie można modyfikować.

C – określa dodatkowe zasady odwoływania się do segmentu programu: dla C=0 odwołanie do segmentu programu jest możliwe tylko w przypadku, gdy CPL=RPL, natomiast dla C=1 dopuszcza się odwołanie do segmentu, gdy CPL>=DPL.

W – bit dopuszczający operacje zapisu dla segmentu danych: dla W=1 zapis dozwolony, W=0 – zabroniony.

E – bit określa położenie segmentu danych względem granicy segmentu. Dla E=0 dane w segmencie zapisuje się powyżej adresu bazowego segmentu („zwykłe" segmenty danych). Dla E=1 dane w segmencie zapisuje się poniżej adresu równego sumie adresu bazowego i rozmiaru segmentu, a segment jest rozszerzalny w dół (ang. expand down). Z sytuacją taką mamy do czynienia w segmentach stosu dla procesorów firmy Intel.

Czterobitowe pole określające typ deskryptora systemowego może przyjmować następujące wartości:

0 – nieokreślone,	1 – dostępny segment TSS procesora 286
2 – tablica LDT,	3 – zajęty segment TSS procesora 286
4 – furtka wywołania procesora 286,	5 – furtka zadania,
6 – furtka przerwania,	7 – furtka potrzasku,
8 – nieokreślona,	9 – dostępny segment TSS (386/486),
A – nieokreślona,	B – zajęty segment TSS (386/486),
C – furtka wywołania,	D – nieokreślona,
E – furtka przerwania (386/486),	F – furtka potrzasku (386/486).

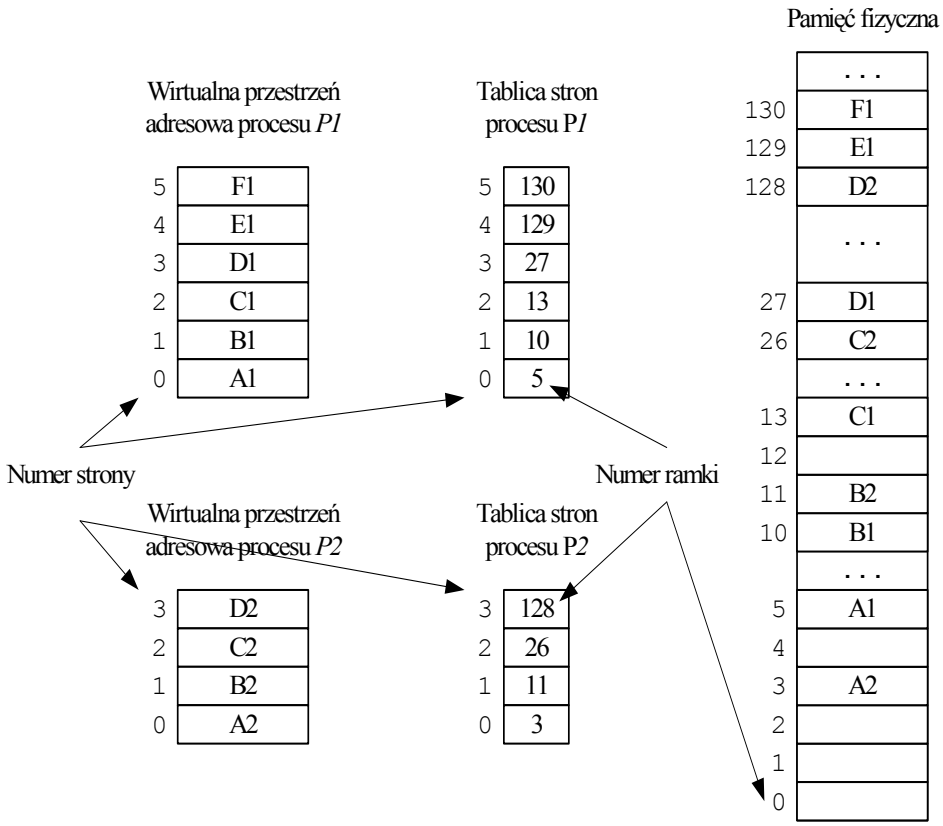
5.4.2. Stronicowanie pamięci

O ile segmentacja pamięci jest stosowana tak w jednozadaniowych, jak i wielozadaniowych systemach operacyjnych, o tyle stronicowanie pamięci stosowane jest wyłącznie w systemach wielozadaniowych. Cechą charakterystyczną takich systemów operacyjnych jest fakt, że w pamięci operacyjnej znajduje się jednocześnie wiele współbieżnie realizowanych procesów, a stronicowanie pamięci pozwala na ich fragmentację w pamięci operacyjnej.

W przypadku stronicowania, wirtualna przestrzeń adresowa każdego procesu jest dzielona na części o jednakowym rozmiarze, określane mianem stron (ang. pages). Analogicznie pamięć fizyczna systemu jest dzielona na części o identycznym rozmiarze jak rozmiary stron, określone mianem ramek (ang. frames). Podczas ładowania obrazu procesu do pamięci system operacyjny odczytuje kolejne strony oraz umieszcza je w aktualnie wolnych ramkach pamięci fizycznej. W celu przechowania informacji o rozmieszczeniu w pamięci fizycznej poszczególnych stron, system operacyjny tworzy strukturę danych określaną mianem tablicy stron, w której do poszczególnych pozycji wpisywane są numery ramek. Są w nich umieszczone odpowiednie strony oraz dodatkowe informacje niezbędne systemowi operacyjnemu do organizacji wymiany stron między pamięcią operacyjną a dyskiem, takie jak znacznik

obecności strony w pamięci, znacznik o modyfikacji strony, znacznik odwołania do strony.

Analogicznie jak w przypadku segmentacji, funkcjonowanie stronicowania pamięci jest zależne od tego, czy suma wirtualnych przestrzeni adresowych wszystkich procesów jest mniejsza od rozmiaru pamięci fizycznej, czy nie. W pierwszym przypadku wszystkie strony wirtualnych przestrzeni adresowych procesów zostają umieszczone w pamięci operacyjnej, natomiast w drugim część stron pozostaje w pamięci dyskowej, podczas gdy do pamięci operacyjnej zostają załadowane strony, które są aktualnie niezbędne do realizacji procesu (rys. 5.21).



Rys. 5.21. Przydział ramek w pamięci fizycznej poszczególnym stronom

Tablice stron poszczególnych procesów są umieszczone w pamięci, a adres tablicy stron jest częścią składową kontekstu procesu. Dzięki temu podczas operacji przełączania kontekstu zostaje odczytany adres tablicy stron procesu, który następnie zostaje załadowany do odpowiedniego rejestru procesora.

Ponieważ wykonywany program zawiera adresy wirtualne, a realizacja programu wymaga wystawiania na magistralę zewnętrzną procesora adresów fizycznych, podczas realizacji programu musi być wykonywana operacja translacji adresów wirtualnych na adresy fizyczne. Operacja ta jest

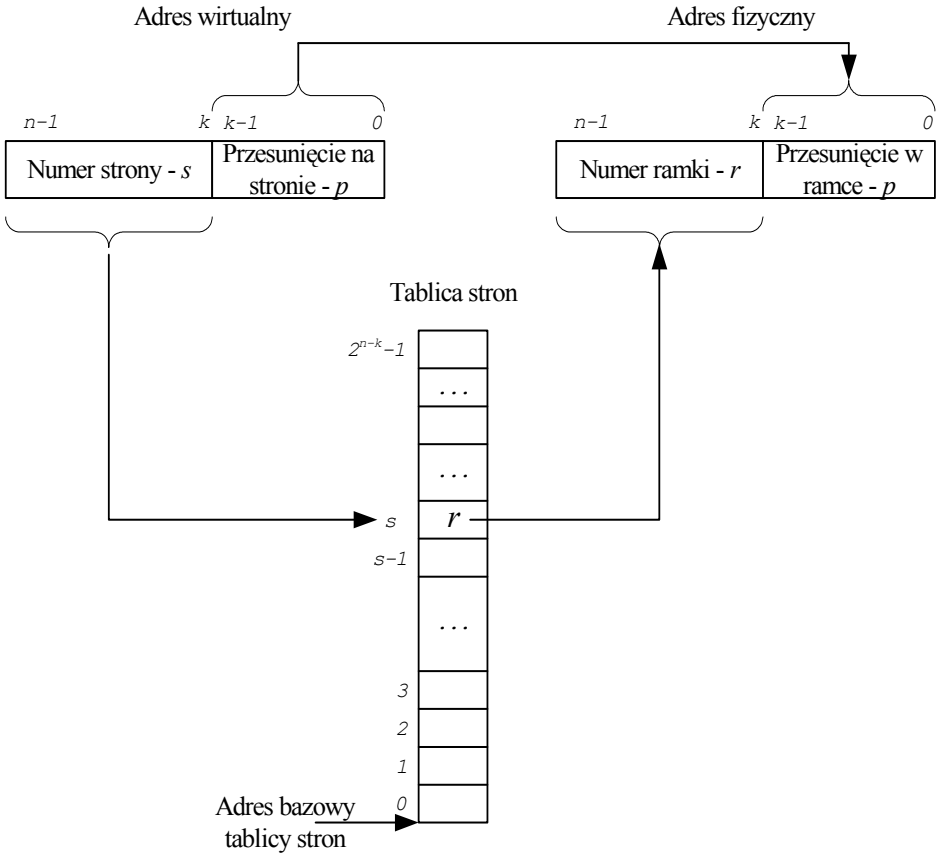
wykonywana sprzętowo przez procesor podczas realizacji każdego rozkazu. Podczas stronicowania pamięci adres wirtualny jest przedstawiony w postaci pary liczb (s, p), gdzie s – numer strony procesu, a p – przesunięcie w zakresie strony. Adres fizyczny zawiera analogicznie parę liczb (r, p), gdzie r – numer ramki, a p – przesunięcie w zakresie ramki, równe przesunięciu w zakresie strony (w pamięci fizycznej umieszczane są całe strony bez przemieszczania bajtów). Oznacza to, że podczas translacji adresu wirtualnego na adres fizyczny należy zamienić numer strony s na numer ramki r. Operacja ta jest szczególnie prosta w przypadku, gdy rozmiar strony jest równy 2^k. Wtedy przesunięcie na stronie/ramce jest reprezentowane przez k młodszych bitów adresu wirtualnego, podczas gdy starsze bity adresu wirtualnego zawierają numer strony. Translacja adresu wirtualnego zawierającego n bitów na n-bitowy adres fizyczny wykonywana jest następująco:

- pobranie starszych n-k bitów adresu wirtualnego, określających numer strony,
- na podstawie adresu bazowego tablicy stron A_{TS} oraz liczby bajtów tworzących jedną pozycję tablicy stron L wyznaczenie adresu potrzebnej pozycji tablicy stron: $A_s=A_{TS}+(n-k)*L$,
- odczyt z tablicy stron numeru ramki r,
- utworzenie adresu fizycznego, w którym n-k starszych bitów określa numer ramki, k młodszych bitów, określające przesunięcie na stronie, jest pobierane bez zmian z adresu wirtualnego.

Algorytm ten został przedstawiony na rys. 5.22. Operacja wyznaczenia adresu fizycznego (wykonywana podczas realizacji każdego rozkazu) wymaga odczytu określonej pozycji tablicy stron, umieszczonej w pamięci operacyjnej. Aby zapewnić dużą szybkość pracy, współczesne procesory są wyposażone w niewielki cache przeznaczony do przechowywania tych pozycji tablicy stron, które są aktualnie wykorzystywane. W przypadku gdy program sięga do strony, której opisu nie ma w cache, procesor pobiera do cache odpowiednią pozycję tablicy stron.

Tablica stron jest strukturą danych o znacznym rozmiarze. Np. dla procesora Pentium, dla którego adres zawiera 32 bity, rozmiar strony wynosi 4 kB = 2^{12}, jedna pozycja tablicy stron zawiera 4 bajty, a liczba możliwych stron wynosi 1 M = 2^{20}, tablica stron zajmuje 4 MB (1 M*4 B). Tablica stron w pamięci wirtualnej stanowi ciągłą strukturę danych o rozmiarze 4 MB, jednak umieszczona w pamięci fizycznej zostaje poddana stronicowaniu, analogicznie jak obrazy procesów. W ten sposób powstaje 2-stopniowy mechanizm stronicowania pamięci stosowany we współczesnych mikroprocesorach.

Rozpatrzmy szczegółowo realizację 2-stopniowego stronicowania pamięci, wykonywaną przez procesor Pentium. Tablica stron o rozmiarze 4 MB zostaje podzielona na 1 k = 2^{10} stron po 4 kB każda. Do określenia numerów ramek, w których umieszczono poszczególne strony tablicy stron, zostaje utworzony katalog stron o rozmiarze 1 strony (1 k pozycji po 4 B każda 1 k*4 B = 4kB).

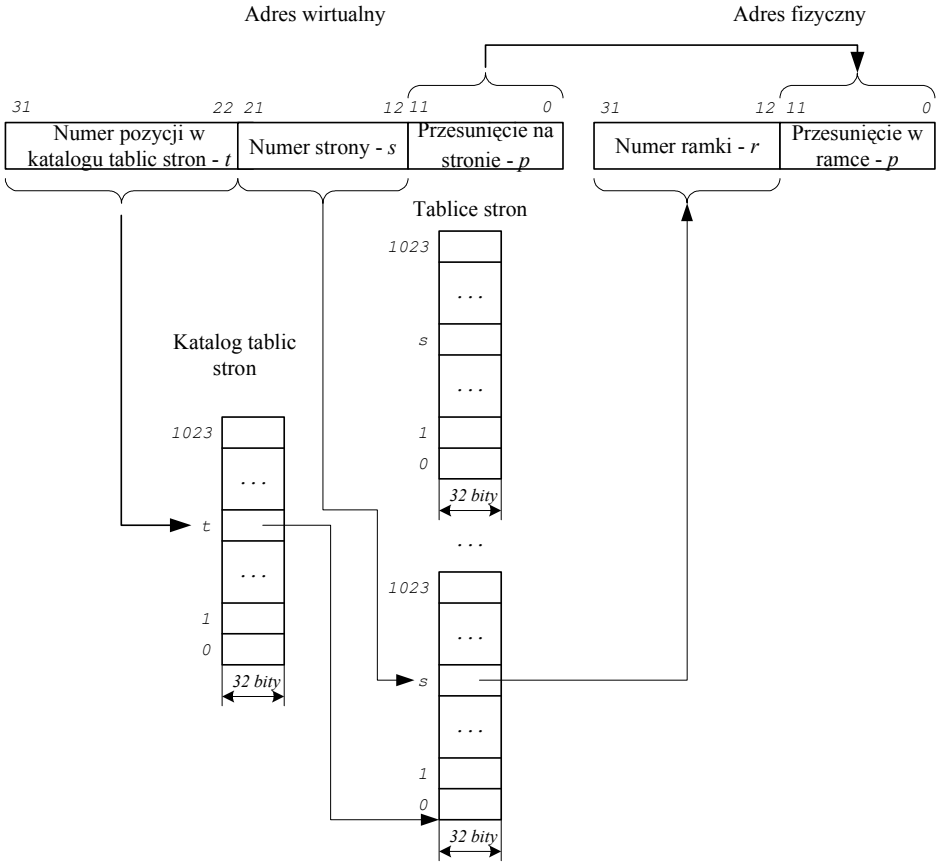


Rys. 5.22. Translacja adresów wirtualnych na fizyczne podczas stronicowania

Dla stronicowania 2-stopniowego adres wirtualny jest reprezentowany przez trójkę liczb (*d, s, p*), gdzie *d* – numer pozycji w katalogu stron, *s* – numer pozycji w tablicy stron określonej przez daną pozycję katalogu stron, *p* – przesunięcie na stronie. Algorytm przekształcenia adresu wirtualnego w adres fizyczny w przypadku stronicowania 2-stopniowego może być przedstawiony następująco:

- na podstawie dziesięciu najstarszych bitów adresowych $A_{31}..A_{22}$ określony zostaje numer pozycji w katalogu stron,
- 4-bajtowa pozycja katalogu stron zawiera adres bazowy konkretnej tablicy stron,
- na podstawie bitów $A_{21}..A_{12}$ określony zostaje numer pozycji w tablicy stron, zawierającej numer ramki,
- najmłodsze 12 bitów adresowych $A_{11}..A_0$ zawiera przesunięcie na stronie, równe przesunięciu w ramce.

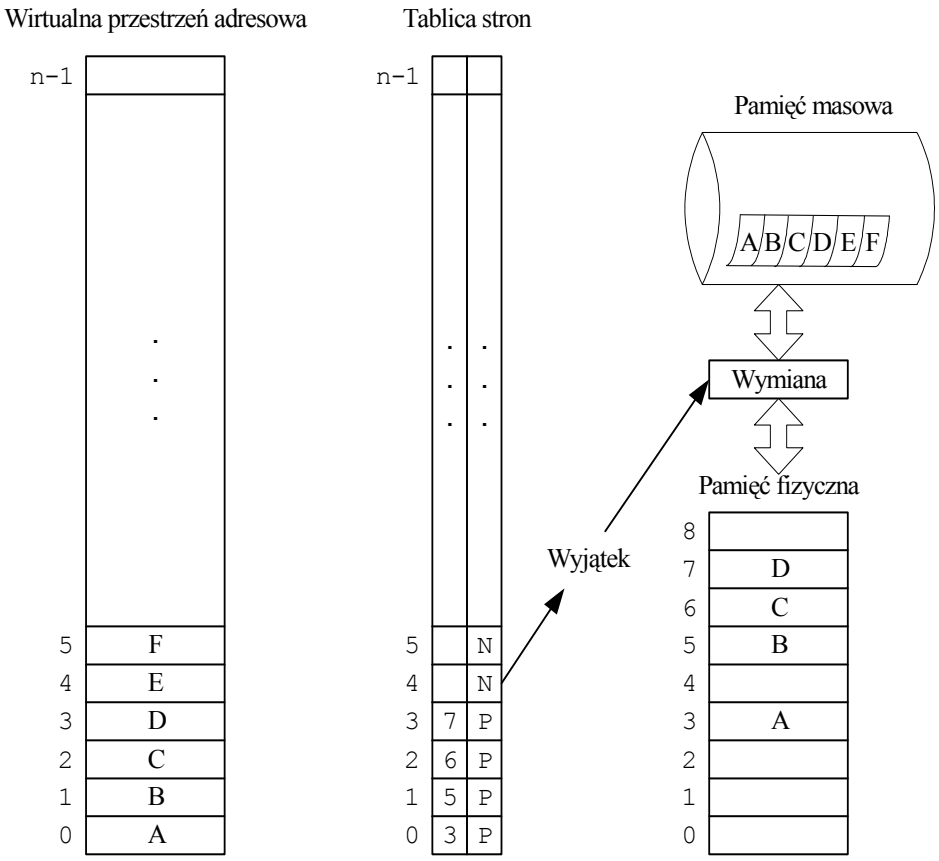
Algorytm przekształcenia adresu wirtualnego na adres fizyczny dla stronicowania 2-stopniowego realizowanego przez procesor Pentium przedstawiono na rys. 5.23.



Rys. 5.23. Translacja adresów wirtualnych na adresy fizyczne realizowana przez procesor Pentium

W przypadku gdy suma wirtualnych przestrzeni adresowych wszystkich obrazów procesów przekracza rozmiar pamięci fizycznej, system operacyjny realizuje operację wymiany stron między pamięcią operacyjną a pamięcią dyskową, wykorzystując znaczniki: obecności strony w pamięci, modyfikacji strony oraz odwołania do strony.

Gdy podczas realizacji rozkazu następuje odwołanie do strony, która aktualnie znajduje się w pamięci (w tablicy stron ustawiony znacznik obecności strony w pamięci), to wykonywana jest operacja przekształcenia adresu wirtualnego na adres fizyczny (rys. 5.24). W przeciwnym wypadku przez procesor jest generowany wyjątek braku strony, a proces przechodzi w stan oczekiwania. Program obsługi wyjątku znajduje na dysku brakującą stronę, a następnie wykonuje operację wymiany strony między pamięcią operacyjną a dyskiem w oparciu o realizowaną przez system operacyjny strategię wymiany stron.



Rys. 5.24. Pamięć wirtualna oparta na stronicowaniu

5.4.3. Segmentacja stronicowana

Ten rodzaj pamięci wirtualnej łączy cechy segmentacji oraz stronicowania pamięci. W tym celu dla każdego procesu system operacyjny tworzy osobną tablicę segmentów (ang. LDT – Local Descriptor Table), w której zapisuje deskryptory poszczególnych segmentów procesu, zawierające: adres bazowy segmentu, rozmiar segmentu oraz prawa dostępu do niego. Jednak adres bazowy segmentu nie jest w tym przypadku adresem fizycznym, lecz liniowym adresem wirtualnym w przestrzeni adresów wirtualnych. W oparciu o te dane układ segmentacji pamięci przetwarza adres wirtualny w adres liniowy, zgodnie z algorytmem segmentacji pamięci.

Podział całej liniowej wirtualnej przestrzeni adresowej procesu na strony oraz pamięci fizycznej na ramki odbywa się analogicznie jak w stronicowaniu pamięci. Poszczególne strony numeruje się w ramach wirtualnej przestrzeni adresowej każdego procesu. System operacyjny tworzy dla każdego procesu tablicę stron, w której przechowuje numery ramek odpowiadające poszczególnym stronom.

Rozdział VI
SYSTEMY PLIKÓW

Jednym z głównych zadań systemu operacyjnego jest sprawna obsługa danych przechowywanych w pamięciach dyskowych. W tym celu system operacyjny zamienia fizyczną strukturę danych na dysku na logiczny model danych wygodny dla użytkownika. Głównymi cechami tego modelu są: drzewiasta struktura katalogów, pliki identyfikowane nazwą, określona lista operacji plikowych. Podstawowym elementem tego modelu jest plik, charakteryzujący się określoną strukturą logiczną oraz fizyczną.

6.1. Cele i zadania systemu plików

Pod pojęciem pliku rozumiemy nazwany obszar pamięci zewnętrznej, do którego można zapisywać dane oraz je odczytywać. Do podstawowych cech plików należy zaliczyć:

- długotrwałe i pewne przechowywanie danych,
- jednoczesne korzystanie z danych przez wiele aplikacji oraz wielu użytkowników.

Użytkownicy oraz aplikacje posiadają możliwości wykonywania szeregu operacji na plikach oraz katalogach, a mianowicie: tworzenie, usuwanie, odczyt oraz modyfikacja danych. Plik zostaje utworzony przez jednego użytkownika (właściciela pliku), jednak mogą z niego korzystać wszyscy użytkownicy w ramach uprawnień przydzielonych przez właściciela. Wszystkie te operacje w systemie operacyjnym realizuje system plików.

W skład systemu plików wchodzą:

- zbiór wszystkich plików w pamięci masowej,
- zbiór struktur danych wykorzystywanych do zarządzania plikami, takich jak: katalogi, deskryptory plików, tablice zawierające informacje o aktualnie wolnych oraz zajętych obszarach dysków,
- zbiór programów systemowych realizujących poszczególne operacje na plikach, takich jak: tworzenie, usuwanie, odczyt, zapis, nadanie nazwy czy poszukiwanie pliku.

System plików udostępnia programom użytkowym zbiór podstawowych operacji na abstrakcyjnym obiekcie, jakim jest plik. Przy tym programista powinien być zwolniony z konieczności zajmowania się szczegółami, takimi jak rozmieszczenie pliku na poszczególnych powierzchniach, ścieżkach i sektorach pamięci dyskowej, buforowanie danych oraz inne niskopoziomowe operacje przesyłania danych. Wszystkie te operacje system plików powinien wykonywać samodzielnie. W ten sposób pełni on rolę pośredniej warstwy, ukrywającej wszystkie szczegóły związane z fizyczną organizacją pamięci masowej, tworzy prosty model logiczny tej pamięci oraz udostępnia dla programów zbiór funkcji pozwalających na manipulowanie plikami.

Zadania realizowane przez system plików są ściśle uzależnione od logicznej organizacji systemu komputerowego. W jednozadaniowym, jednostanowiskowym systemie operacyjnym (np. MS-DOS) system plików realizuje następujące zadania:

- identyfikacja plików poprzez nazwy,
- realizacja interfejsu programowego dla programów użytkowych,
- przeniesienie logicznego modelu systemu plików na fizyczną organizację pamięci masowej,
- zapewnienie odporności systemu plików na zaniki napięcia zasilania oraz błędne funkcjonowanie sprzętu i oprogramowania.

Zadania systemu plików komplikują się w wielozadaniowych, jednostanowiskowych systemach operacyjnych, które mimo że są przeznaczone dla jednego użytkownika, dają mu możliwość jednoczesnego uruchomienia wielu aplikacji, które mogą wykorzystywać wspólne pliki. W takim przypadku plik staje się zasobem współdzielonym, a system operacyjny musi rozwiązywać wiele z tym związanych problemów. W szczególności, w systemie plików muszą być przewidziane środki blokowania całych oraz części plików, zapobiegania wyścigom, wykluczania blokad itp.

W wielozadaniowych systemach operacyjnych pojawia się jeszcze jedno zadanie: ochrona plików jednego użytkownika przed nieupoważnionymi operacjami innych użytkowników, natomiast w sieciowych systemach operacyjnych powstaje wiele nowych zadań omówionych szczegółowo w rozdziale 8.

6.1.1. Rodzaje plików

Systemy plików obsługują różne typy plików, do których można zaliczyć: pliki zwykłe, katalogowe (katalogi), specjalne, potoki nazwane, dowiązania symboliczne itp.

Pliki zwykłe lub po prostu *pliki*, zawierają dowolne dane. Większość współczesnych systemów operacyjnych (UNIX, Windows, OS) nie wprowadza ograniczeń i nie kontroluje ani zawartości, ani struktury plików zwykłych. Zawartość pliku zwykłego jest określona przez aplikację współpracującą z plikiem. Głównymi plikami zwykłymi są pliki tekstowe, zawierające zbiory bajtów reprezentujących znaki alfanumeryczne; pliki binarne, zawierające bajty danych, którym nie przypisuje się żadnego znaczenia, a także szczególny rodzaj plików binarnych – pliki wykonywalne, zawierające obrazy procesów.

Katalogi (pliki katalogowe) – to szczególny rodzaj plików zawierających informacje o plikach zgrupowanych przez użytkownika. Można powiedzieć, że plik katalogowy zawiera listę plików oraz podkatalogów przypisanych do danego katalogu. W skład katalogu mogą wchodzić pliki dowolnego typu, w tym także inne katalogi, dzięki czemu tworzona jest drzewiasta struktura katalogów. Katalogi określają związek między nazwami plików a ich charakterystykami, wykorzystywanymi przez system plików do zarządzania

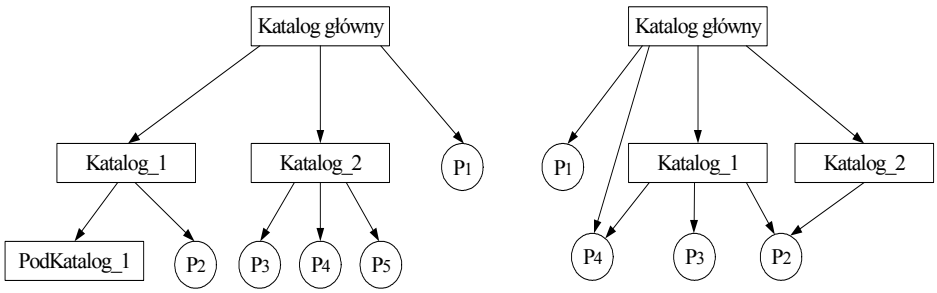
nimi. Do takich charakterystyk można zaliczyć informacje o typie pliku, rozmieszczeniu pliku w pamięci masowej, prawach dostępu do pliku oraz datach utworzenia, ostatniej modyfikacji, ostatnim dostępie. Dostęp do plików katalogowych jest możliwy jedynie za pośrednictwem właściwych funkcji systemowych wykonujących odpowiednie operacje na tych plikach. Wykonywanie operacji na plikach katalogowych z zastosowaniem programów służących do operacji na nich jest niemożliwy, gdyż mógłby doprowadzić do uszkodzenia systemu plików.

Pliki specjalne – to pliki fikcyjne związane z poszczególnymi urządzeniami WE/WY, służące do unifikacji mechanizmów dostępu do plików i urządzeń WE/WY. Pliki specjalne pozwalają użytkownikowi na wykonywanie operacji WE/WY przy pomocy zwykłych poleceń zapisu oraz odczytu z pliku. Polecenia te w pierwszej kolejności są przetwarzane przez system plików, a następnie na określonym etapie wykonywania poleceń system operacyjny przetwarza je w polecenia zarządzania odpowiednimi urządzeniami WE/WY.

Współczesne systemy plików stosują także inne rodzaje plików, takie np. jak potoki nazwane, dowiązania symboliczne - omówione w dalszej części.

6.1.2. Hierarchiczna struktura systemu plików

Dzięki temu, że w skład katalogu mogą wchodzić katalogi niższego poziomu (podkatalogi), system plików posiada strukturę hierarchiczną. Graf opisujący hierarchię katalogów może mieć strukturę drzewa lub sieci. Katalogi tworzą drzewo, gdy dany plik może wchodzić w skład wyłącznie jednego katalogu (rys. 6.1), oraz sieć – gdy plik może być przypisany do wielu katalogów. W systemach operacyjnych MS-DOS oraz Windows katalogi tworzą strukturę drzewiastą, natomiast w systemie UNIX – sieciową.



Rys. 6.1. Hierarchia systemu plików

6.1.3. Montowanie systemu plików

W ogólnym przypadku system komputerowy może zawierać szereg urządzeń dyskowych. Nawet typowy komputer PC zwykle posiada jeden napęd dysków twardych, jeden napęd dysku elastycznego oraz jeden napęd dysku optycznego. Komputery profesjonalne są z zasady wyposażone w wiele

napędów dyskowych. Ponadto jeden napęd dyskowy może być podzielony za pomocą oprogramowania systemowego na szereg dysków logicznych (partycji).

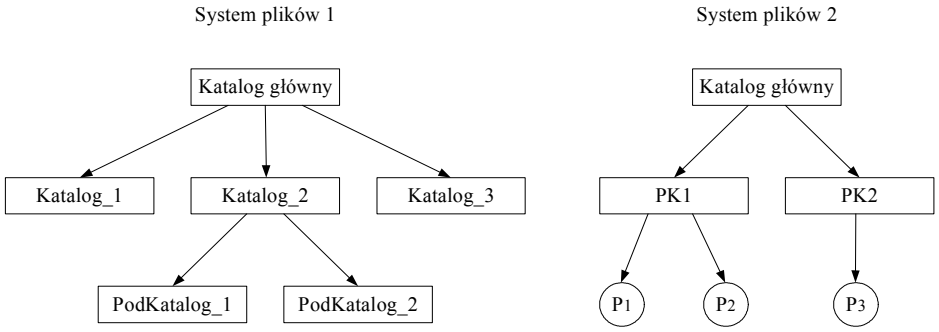
Zarządzanie zbiorem pamięci masowych danego komputera może odbywać się w dwojaki sposób:

- dla każdego urządzenia dyskowego (fizycznego lub logicznego) jest tworzony odrębny system plików,
- w systemie operacyjnym jest tworzony jeden system plików, obejmujący wszystkie urządzenia dyskowe.

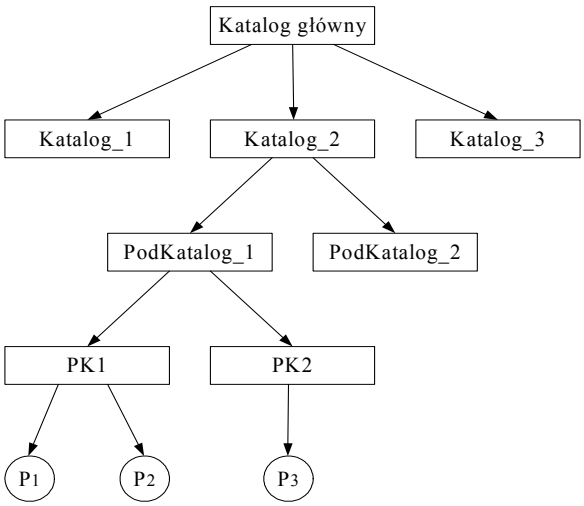
W pierwszym przypadku każdy z systemów plików związanych z poszczególnymi urządzeniami dyskowymi posiada swoją nazwę logiczną. W systemach MS-DOS oraz Windows poszczególne systemy plików posiadają nazwy w postaci jednej litery alfabetu łacińskiego wraz z dwukropkiem (np. a:, c: itd.).

W drugim przypadku operacja łączenia wszystkich systemów plików w jeden system, reprezentowany przez jedną hierarchiczną strukturę katalogów, określana jest mianem *montowania*. Rozpatrzmy w jaki sposób jest realizowana ta operacja w systemach UNIX.

Spośród wszystkich dysków logicznych występujących w systemie, system operacyjny wyróżnia jeden, określany mianem systemowego. Na rys. 6.2 przedstawiono dwa systemy plików, umieszczone na dwu różnych dyskach logicznych. W celu połączenia obydwu systemów plików w strukturze katalogów dysku systemowego zostaje wybrany pewien (dowolny) katalog (na rys. 6.2 katalog `PodKatalog_1`), który po wykonaniu operacji montowania staje się katalogiem głównym dla drugiego systemu plików. Innymi słowy, poprzez katalog `PodKatalog_1` drugi system plików zostaje domontowany do ogólnego drzewa katalogowego (rys. 6.3).



Rys. 6.2. Dwa systemy plików na dwu różnych dyskach logicznych



Rys. 6.3. System plików po wykonaniu operacji montowania

Po wykonaniu operacji montowania systemu plików system ten jest reprezentowany przez jedno drzewo katalogów, a użytkownik nie zdaje sobie sprawy, na którym dysku fizycznym (logicznym) znajduje się dany plik.

6.1.4. Atrybuty plików

Pojęcie pliku zawiera w sobie nie tylko dane oraz nazwę pliku, lecz także atrybuty, czyli dodatkowe informacje opisujące właściwości pliku. Do głównych atrybutów należy zaliczyć:

- typ pliku (np. plik zwykły, katalog, plik specjalny itp.),
- właściciela pliku,
- twórcę pliku,
- hasło dostępu do pliku,
- prawa dostępu do pliku,
- czas i datę stworzenia, ostatniego dostępu oraz ostatniej modyfikacji pliku,
- aktualny rozmiar pliku,
- maksymalny rozmiar pliku,
- znaczniki:
    - „tylko do odczytu”,
    - „plik ukryty”,
    - „plik systemowy”,
    - „plik archiwalny”,
    - „plik tymczasowy”,
    - „plik binarny/znakowy”,
    - blokady.

W systemach operacyjnych stosowane są różne podzbiory wymienionych wyżej atrybutów. Atrybuty plików mogą być zapisane w dwojaki sposób:

- w pliku katalogowym, w którym znajdują się opisy plików przypisanych do danego katalogu (np. MS-DOS),
- w specjalnych strukturach danych, przechowywanych w odrębnych tablicach. W takim przypadku w pliku katalogowym znajdują się odwołania do odpowiednich pozycji tej tablicy (np. UNIX).

Rekord pliku katalogowego w systemie operacyjnym MS-DOS zawiera następujące pola:

- nazwa pliku,
- rozszerzenie nazwy,
- znaczniki,
- czas,
- data,
- numer pierwszego klastra,
- rozmiar.

W systemie operacyjnym UNIX zbiór atrybutów pliku jest umieszczony w strukturze określonej mianem węzła identyfikacyjnego pliku (ang. I-node), a zbiór wszystkich I-nodów jest umieszczony w tablicy I-nodów. Struktura pliku katalogowego jest w tym przypadku bardzo prosta i zawiera dwa pola: numer I-noda oraz nazwa pliku. Oddzielenie nazwy pliku od jego atrybutów zwiększa elastyczność systemu plików. Np. jeden plik fizyczny może być umieszczony w wielu katalogach, pod różnymi nazwami (są to tzw. dowiązania twarde – ang. hard links).

6.1.5. Logiczna organizacja pliku

W ogólnym przypadku dane przechowywane w plikach posiadają określoną strukturę, która jest podstawą opracowania programów przetwarzania danych przechowywanych w plikach. Aby tekst mógł być prawidłowo wyprowadzony na ekran, program musi mieć możliwość wydzielenia poszczególnych słów, zdań, akapitów itp. Znacznikami oddzielającymi poszczególne elementy mogą być określone kombinacje znaków. Inną możliwością wydzielenia poszczególnych elementów pliku może być określenie adresów dla poszczególnych elementów pliku względem początku pliku. Definiowanie struktury danych w pliku może być realizowane przez system operacyjny lub może być przekazane do aplikacji klienckiej.

W przypadku gdy wszystkie działania związane z określeniem struktury oraz interpretacją zawartości pliku są w całości przekazane do aplikacji klienckiej, plik stanowi ciąg bajtów i nie posiada żadnej narzuconej struktury. W celu wykonania określonej operacji na danych, aplikacja kliencka pobiera dane z pliku używając standardowych funkcji systemowych (np. podając przesunięcie względem początku pliku oraz liczbę bajtów), a następnie interpretuje pobrane dane zgodnie z zadanym w programie algorytmem. Format

danych w pliku znany jest jedynie aplikacji klienckiej, natomiast system operacyjny traktuje plik wyłącznie jako ciąg bajtów. Taki model pliku, zastosowany w systemie operacyjnym UNIX, stosowany jest we wszystkich współczesnych systemach operacyjnych.

Innym modelem pliku, stosowanym w wielu dawnych systemach operacyjnych (OS/360, DEC RSX, VMS), lecz praktycznie niestosowanym współcześnie, jest plik z określoną strukturą. W takim przypadku utrzymywanie struktury pliku jest realizowane przez system plików, który „widzi" plik jako uporządkowany ciąg rekordów i wszystkie operacje wykonuje na poziomie poszczególnych rekordów. System plików musi posiadać informacje o strukturze plików, niezbędne dla wydzielenia z pliku poszczególnych rekordów. System plików umożliwia aplikacji klienckiej dostęp do określonego rekordu, podczas gdy operacje przetwarzania danych zawartych w rekordzie są wykonywane przez aplikację kliencką. We współczesnych systemach informatycznych operacje przetwarzania plików o określonej strukturze są realizowane przez systemy zarządzania bazami danych. Elementarną strukturą danych, jaką może operować programista aplikacji klienckiej, jest w tym przypadku logiczny rekord, będący elementem pliku.

Systemy operacyjne mogą stosować dwa sposoby dostępu do rekordów w pliku:

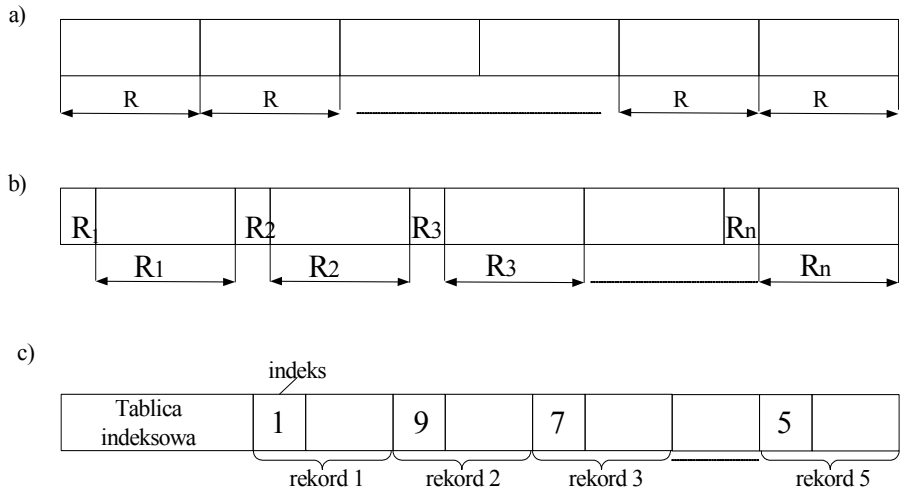
- odczyt lub zapis rekordów w sposób sekwencyjny (pliki z dostępem sekwencyjnym),
- pozycjonowanie rekordów w pliku - przed operacją odczytu lub zapisu poszukiwany jest rekord o określonym numerze (pliki z dostępem swobodnym).

Pliki, w których rekordy posiadają stałą długość, mogą być plikami o dostępie sekwencyjnym oraz swobodnym. W przypadku dostępu sekwencyjnego poszczególne rekordy umieszczone w pliku mogą być wyłącznie odczytywane kolejno, poczynając od początku pliku, bez możliwości zmiany tej kolejności. Aby więc odczytać rekord o numerze *N*, należy najpierw odczytać wszystkie *N-1* rekordów początkowych. Natomiast gdy plik charakteryzuje się dostępem swobodnym, w celu odczytu rekordu o numerze *N* wyznaczany jest najpierw adres początku tego rekordu w pliku, równy *(N-1)*R* (R – długość rekordu), a następnie wykonywana jest operacja odczytu tego rekordu (rys. 6.4a). Pliki z dostępem swobodnym charakteryzują się znacznie krótszym czasem dostępu do danych niż pliki z dostępem sekwencyjnym.

Pliki zawierające rekordy o zmiennej długości mogą być wyłącznie plikami o dostępie sekwencyjnym, gdyż nie ma w takim przypadku możliwości określenia adresu początkowego rekordu o podanym numerze (rys. 6.4b).

Szczególnym typem plików są pliki indeksowane. Ten sposób dostępu do danych przechowywanych w pliku zapewnia bardzo dużą szybkość oraz może być stosowany dla plików o stałej i zmiennej długości rekordu. W takim przypadku z plikiem danych zostaje związana tablica indeksowa, będąca tablicą

2-kolumnową, w której zapisywany jest klucz indeksowy jednoznacznie opisujący poszczególne rekordy oraz adres rekordu o podanym kluczu (rys. 6.4c). Dzięki takiej organizacji, podczas poszukiwania określonego rekordu w pliku najpierw należy przeszukać tablicę indeksową (o niewielkim rozmiarze w stosunku do rozmiaru pliku) w celu wyznaczenia adresu danego rekordu, a następnie wykonać odpowiednią operację odczytu lub zapisu rekordu.



Logiczna organizacja indeksu

Indeks	1	2	3		
Adres	35	237	423		

Rys. 6.4. Różne logiczne organizacje plików

W systemach operacyjnych można się także spotkać z indeksowo-sekwencyjną organizacją plików. Polega ona na tym, że w tablicy indeksowej podany jest adres początkowy pewnej sekwencji rekordów. W takim przypadku podczas poszukiwania określonego rekordu należy w pierwszej kolejności sięgnąć do tablicy indeksowej i wyznaczyć adres początkowy sekwencji rekordów, a następnie sekwencyjnie poszukiwać określonego rekordu.

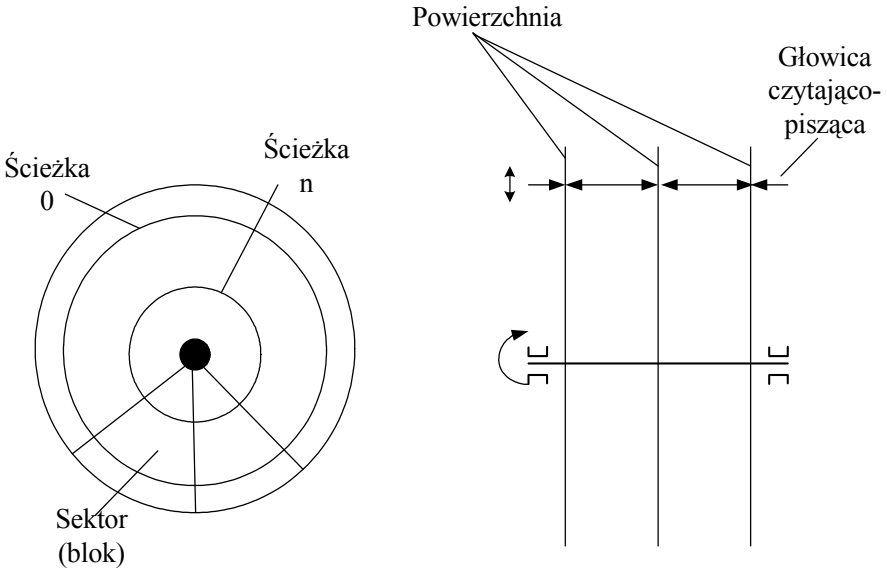
6.2. Fizyczna organizacja systemu plików

Logiczny obraz systemu plików w postaci hierarchicznej struktury katalogów posiada niewiele wspólnego z fizycznym zapisem danych na dysku. Plik, widziany przez użytkownika jako ciągły zbiór bajtów, w rzeczywistości

zostaje podzielony na określone porcje, które są zapisane w różnych miejscach pamięci dyskowej, a pliki logicznie zgrupowane w jednym katalogu w rzeczywistości mogą być rozrzucone po różnych powierzchniach, ścieżkach i sektorach pamięci dyskowej. Fizyczna organizacja dysku określa zasady rozmieszczenia plików, katalogów oraz danych systemowych w pamięci dyskowej.

6.2.1. Dyski, partycje, sektory, klastry

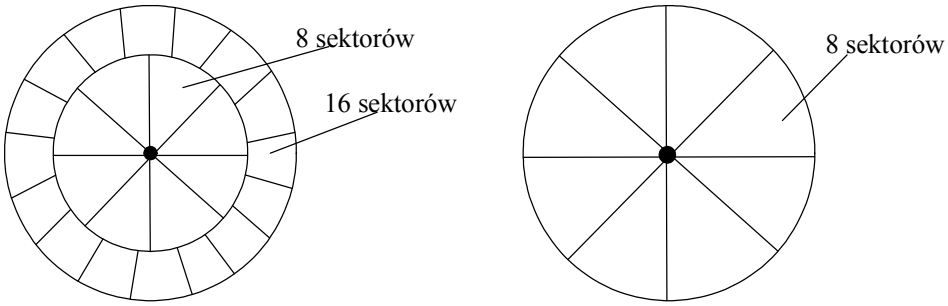
Podstawowym typem pamięci masowych, stosowanym powszechnie we współczesnych systemach komputerowych, są dyski magnetyczne. Składają się one z jednego (lub kilku) szklanych, metalowych lub plastikowych krążków, pokrytych obustronnie materiałem magnetycznym (rys. 6.5). Na poszczególnych powierzchniach krążków (talerzy) umieszczone są koncentryczne okręgi (ścieżki – ang. tracks), na których zapisywane są dane. Liczba ścieżek zależy od typu dysku, a ich numeracja zaczyna się od ścieżki zewnętrznej o numerze 0. Odczyt i zapis danych na poszczególnych ścieżkach dysku odbywa się podczas obrotu dysku, przy pomocy głowicy magnetycznej. Z każdą powierzchnią pamięci dyskowej związana jest jedna głowica, a wszystkie głowice przemieszczają się wspólnie między ścieżkami.



Rys. 6.5. Budowa dysku twardego

Zbiór ścieżek o tym samym numerze na wszystkich powierzchniach dyskowych określa się mianem cylindra. Każda ścieżka zostaje podzielona na fragmenty określane sektorami (ang. sector). Liczba sektorów na poszczególnych ścieżkach może być jednakowa lub zmienna. W pierwszym przypadku, ze względu na mniejszą długość liniową, ścieżki wewnętrzne

charakteryzują się większą gęstością zapisu danych. Natomiast w drugim przypadku liczba sektorów na ścieżkach zewnętrznych jest większa niż na ścieżkach wewnętrznych, co daje możliwość zrównania gęstości zapisu danych (rys. 6.6).



Rys. 6.6. Podział ścieżek na sektory

Liczba bajtów danych przechowywanych w jednym sektorze jest stała i wynosi najczęściej 0,5 kB = 512 B. Jest to minimalna jednostka danych wymienianych między pamięcią operacyjną a pamięcią dyskową, posiadająca trójskładnikowy adres: numer powierzchni (głowicy), numer cylindra (ścieżki) oraz numer sektora. Poza danymi w sektorze zapisane są także adresy oraz ciągi bajtów służące do synchronizacji sterownika dyskowego z obracającym się dyskiem. Ścieżki oraz sektory na poszczególnych powierzchniach pamięci dyskowej są tworzone podczas operacji formatowania niskopoziomowego. Informacje te są następnie wykorzystywane przez sterownik dyskowy, natomiast nie są istotne dla systemu operacyjnego.

System operacyjny tworzy logiczną jednostkę danych na dysku, będącą wielokrotnością sektora, określaną mianem klastra (ang. cluster) $K = 2^n * S$ ($n=0,1,2,...$). Klaster może więc zawierać: 1 sektor ($n=0$), 2 sektory ($n=1$), 4 sektory ($n=2$) itd. Przygotowanie dysku do określonego systemu plików jest wykonywane podczas operacji formatowania wysokopoziomowego, w czasie której wykonuje się następujące operacje:

- określenie rozmiaru klastra,
- zapis na dysku informacji niezbędnych dla systemu plików (np. wolna i zajęta przestrzeń pamięci dyskowej, rozmieszczenie plików i katalogów na dysku itd.),
- zapis oprogramowania ładującego system operacyjny podczas startu systemu.

Dysk fizyczny może zostać podzielony na części logiczne, określane mianem partycji. Są to ciągłe obszary dysku fizycznego, które są przez system operacyjny traktowane jako odrębne dyski logiczne. Na każdym dysku logicznym może funkcjonować jeden system plików, natomiast na poszczególnych partycjach mogą być zainstalowane różne systemy plików, współpracujące z różnymi systemami operacyjnymi.

6.2.2. Organizacja fizyczna oraz adresowanie pliku

Fizyczna organizacja pliku określa sposób rozmieszczenia pliku na dysku. Do podstawowych kryteriów efektywności fizycznej organizacji plików należy zaliczyć:

- szybkość dostępu do danych,
- objętość informacji adresowej pliku,
- stopień fragmentacji przestrzeni dyskowej,
- maksymalny rozmiar pliku.

Można wyróżnić następujące cztery sposoby zapisu informacji o rozmieszczeniu pliku na dysku, stosowane w systemach operacyjnych:

- ciągłe rozmieszczenie plików na dysku,
- jednokierunkowa lista klastrów,
- jednokierunkowa lista indeksów,
- spis klastrów.

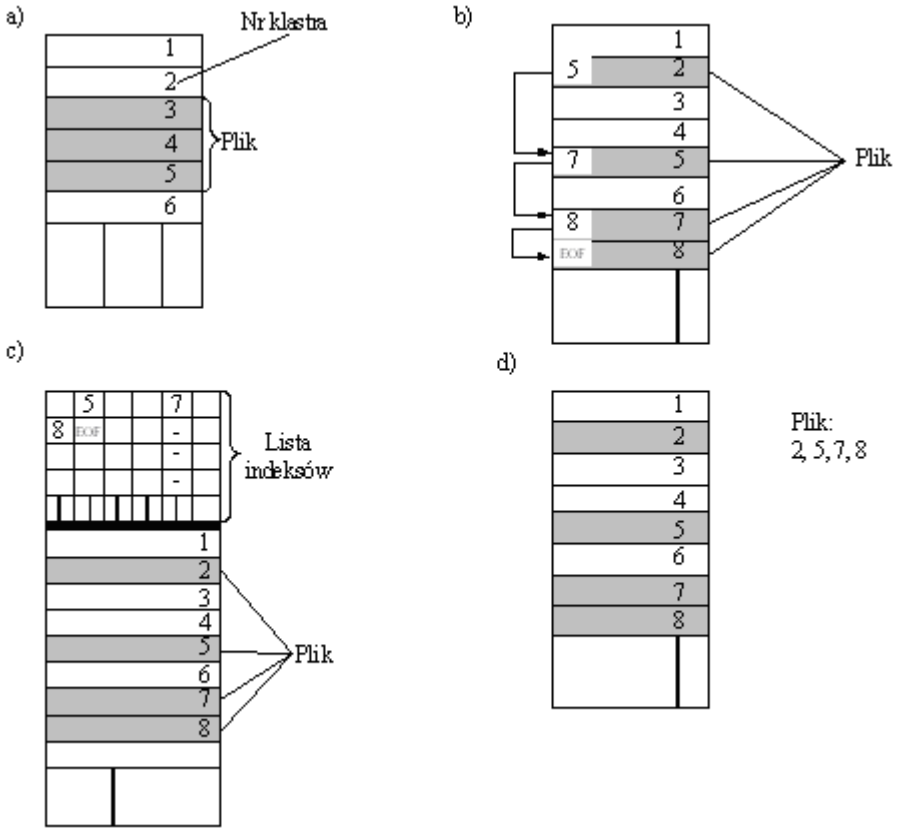
Ciągłe rozmieszczenie pliku na dysku (rys. 6.7a) charakteryzuje się istotnymi zaletami, a przede wszystkim: szybkim dostępem do danych w pliku oraz minimalnym rozmiarem pamięci niezbędnym do zapisu informacji adresowej (wystarczy zapisać numer pierwszego klastra oraz liczbę klastrów). Zastosowanie jednak tego modelu jest praktycznie niemożliwe, gdyż przy zmieniających się rozmiarach plików oraz usuwaniu jednych i tworzeniu nowych plików należałoby przemieszczać pliki na dysku w celu łączenia wolnych obszarów, co znacznie spowolniłoby pracę pamięci dyskowej.

Jednokierunkowa lista klastrów (rys. 6.7b) umożliwia zapis na dysku plików pofragmentowanych, co pozwala umieszczać fragmenty pliku w aktualnie wolnych klastrach. Przy takiej fizycznej organizacji pliku, na początku każdego klastra zostaje umieszczony wskaźnik na kolejny klaster, w którym znajduje się ciąg dalszy pliku, a w przypadku gdy jest to ostatni klaster pliku – znacznik końca pliku. Model ten charakteryzuje się powolnym dostępem do danych, wynikającym z sekwencyjnego dostępu do poszczególnych klastrów. Jeżeli np. chcemy odczytać piąty w kolejności klaster pliku, to niezbędny jest sekwencyjny odczyt wszystkich wcześniejszych klastrów, gdyż jest w nich zapisana informacja adresowa o umieszczeniu kolejnego klastra. Informacja o pierwszym klastrze pliku (początek pliku) jest umieszczona w pliku katalogowym, zawierającym spis plików przypisanych do danego katalogu.

Jednokierunkowa lista indeksów (rys. 6.7c) jest modyfikacją poprzedniej metody, polegającą na tym, że informacja o numerze kolejnego klastra pliku zostaje umieszczona poza plikiem, w odrębnej systemowej strukturze danych, określanej mianem tablicy rozmieszczenia (alokacji) plików (ang. FAT - File Allocation Table). Liczba pozycji tablicy alokacji plików jest równa liczbie klastrów na danym dysku. Każda pozycja FAT zawiera informację o numerze kolejnego klastra wchodzącego w skład danego pliku, lub specjalną wartość

oznaczającą koniec pliku, a wartość równa zero oznacza, że klaster jest wolny. Taka organizacja posiada właściwości analogiczne do poprzedniej, jednak cechuje się znacznie szybszym dostępem do danych. Wynika to z faktu, że w celu dostępu do entego klastra pliku nie trzeba sekwencyjnie odczytywać kolejnych klastrów poczynając od początku pliku, lecz wystarczy odczytać kolejne pozycje FAT.

Ostatni sposób zapisu informacji o rozmieszczeniu pliku na dysku polega na stworzeniu spisu numerów klastrów wchodzących w skład danego pliku. Zaletą takiego podejścia jest duża szybkość dostępu do danych zapisanych w dowolnym klastrze (bezpośredni dostęp do poszczególnych klastrów bez konieczności przeglądania FAT). Jednak model ten charakteryzuje się istotną wadą polegającą na tym, że spis numerów klastrów zależy od długości pliku i dla dużych plików może zajmować bardzo wiele miejsca. W celu zmniejszenia rozmiaru listy klastrów, dla dużych plików wprowadza się pośredni sposób adresowania klastrów wchodzących w skład pliku. Taki sposób zapisu informacji o rozmieszczeniu pliku na dysku jest stosowany w systemach plików UNIX System V oraz UFS.



Rys. 6.7 Fizyczna organizacja pliku: rozmieszczenie ciągłe (a), jednokierunkowa lista klastrów (b), jednokierunkowa lista indeksów (c), spis klastrów (d)

Rozpatrzmy zapis informacji o rozmieszczeniu pliku w UNIX system V, dla przypadku gdy klaster zawiera dwa sektory (2^1*0,5 kB = 1 kB). Informacja o rozmieszczeniu pliku jest zawarta w 13-pozycyjnej tablicy indeksowej, wchodzącej w skład węzła identyfikacyjnego pliku. Każda pozycja tablicy indeksowej jest liczbą 32-bitową, określającą numer klastra. Liczba klastrów jest więc równa 2^{32} = 4 G, co przy rozmiarze klastra 1 kB oznacza, że maksymalny rozmiar dysku wynosi 2^{42} = 4 TB. Pierwsze dziesięć pozycji tablicy indeksowej zawiera pozycje bezpośrednie, co oznacza, że pozycje te zawierają numery klastrów, w których umieszczony jest plik. Ponieważ pozycji bezpośrednich jest 10, więc maksymalny rozmiar pliku opisany w ten sposób wynosi 10 kB (rys. 6.8). Pozycja nr 10 opisuje rozmieszczenie pliku w sposób pośredni, gdyż zawiera numer klastra, w którym zapisana jest tablica indeksowa. Ponieważ rozmiar klastra wynosi 1 kB, a jedna pozycja tablicy zawiera 4 B (32 bity), więc w klastrze jest zawarte 256 pozycji tablicy indeksowej. Oznacza to, że maksymalny rozmiar pliku opisanego w ten sposób wynosi 256 kB. Pozycja nr 11 opisuje rozmieszczenie pliku w sposób podwójny pośredni, co oznacza, że zawiera numer klastra, w którym znajduje się tablica, a każda pozycja tej tablicy zawiera numer klastra zawierającego część właściwej tablicy indeksowej. Maksymalny rozmiar pliku opisanego w ten sposób wynosi 256^2 * 1 kB = 64 MB. Ostatnia (nr 12) pozycja tablicy indeksowej opisuje rozmieszczenie pliku w sposób potrójny pośredni (rys. 6.8). Maksymalny rozmiar pliku wynosi w tym przypadku 256^3 * 1 kB = 16 GB.

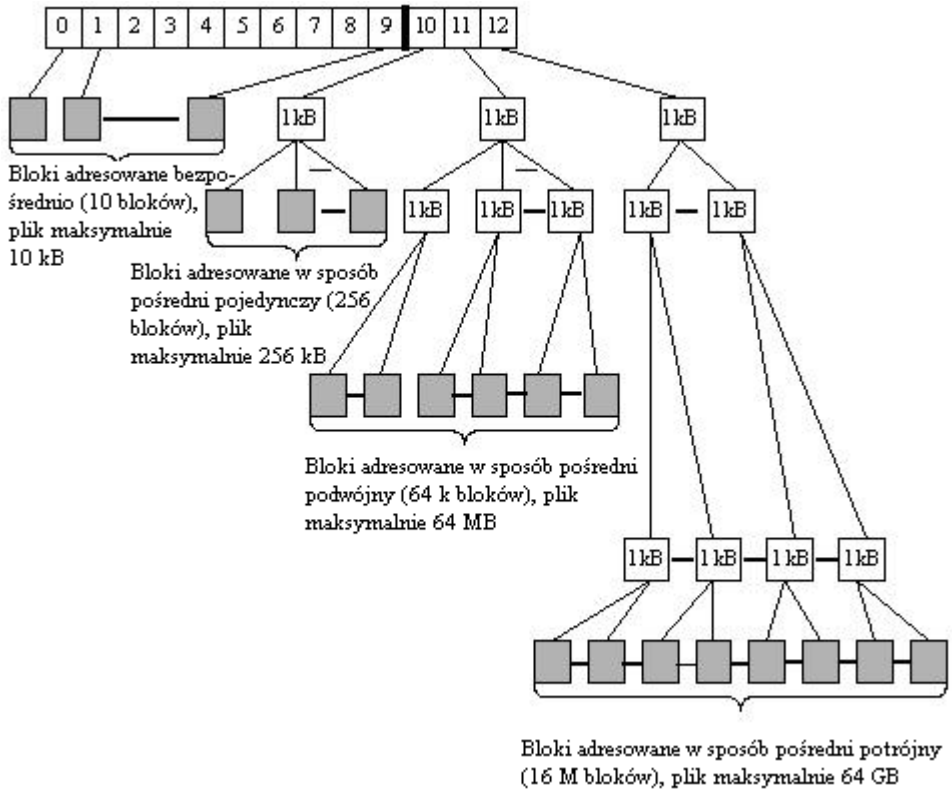
Metoda zapisu informacji o rozmieszczeniu pliku na dysku w postaci spisu numerów klastrów wchodzących w skład pliku jest także stosowana, w nieco zmodyfikowanej formie, w systemie NTFS. System NTFS wprowadza pojęcie ekstentu (ang. extent), który oznacza zbiór kolejnych klastrów zajętych przez dany plik. W celu określenia ekstentu wystarczy podać początkowy numer klastra oraz liczbę klastrów wchodzących w skład ekstentu. Upraszcza to znacznie tablicę indeksową z UNIX system V, gdyż nie trzeba wyszczególniać wszystkich klastrów wchodzących w skład pliku, a jedynie ekstenty. W przypadku niewielkiej fragmentacji pliku na dysku liczba ekstentów może być niewielka, nawet dla dużych plików. W szczególności nawet bardzo duży plik może zawierać jeden ekstent (w przypadku gdy zajmuje kolejne klastry).

6.2.3. Fizyczna organizacja FAT

Partycja dyskowa sformatowana dla systemu FAT zawiera następujące bloki danych (rys. 6.9):

- sektor BOOT zawierający program początkowego ładowania systemu operacyjnego,
- tablicę alokacji plików FAT zawierającą informacje o rozmieszczeniu plików i katalogów na dysku,
- kopię FAT,

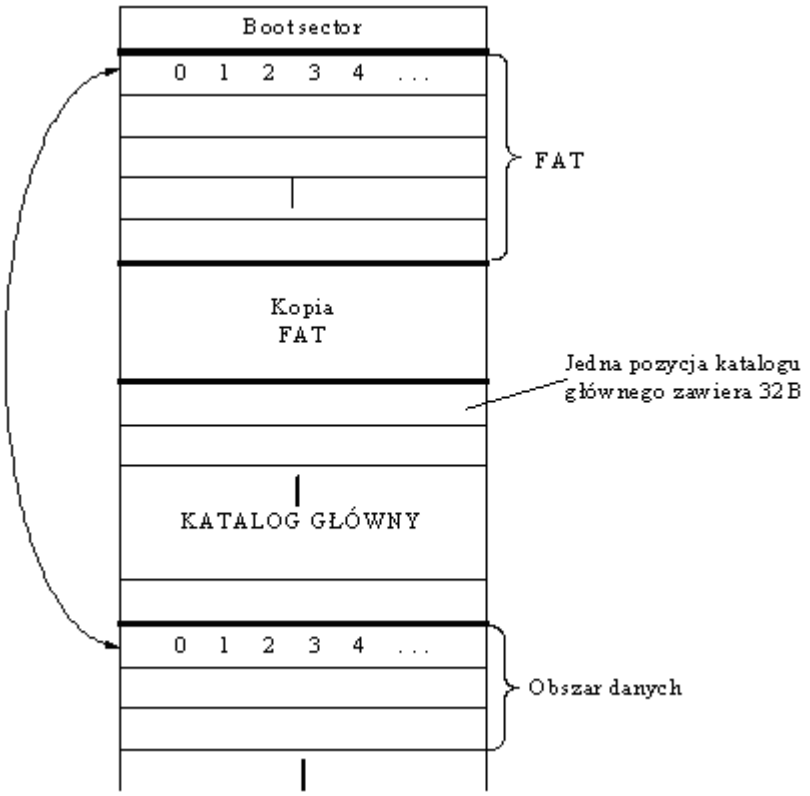
- katalog główny partycji umieszczony w ściśle określonym obszarze dysku o rozmiarze 32 sektorów (16 kB), co pozwala na zapis 512 pozycji (plików i katalogów) umieszczonych w katalogu głównym (każda pozycja katalogu głównego zajmuje 32 B),
- obszar danych przeznaczony do zapisu wszystkich plików oraz katalogów z wyjątkiem katalogu głównego.



Rys. 6.8. Zapis informacji o rozmieszczeniu pliku w UNIX system V

System plików FAT obsługuje dwa rodzaje plików: pliki zwykłe oraz katalogi. FAT jest tablicą o liczbie elementów równej liczbie klastrów w partycji. Pomiędzy klastrami wchodzącymi w skład partycji a pozycjami tablicy FAT funkcjonuje wzajemnie jednoznaczna zależność (zerowa pozycja w tablicy FAT jest związana z klastrem nr 0 itd.). Pozycja tablicy FAT może zawierać następujące wartości, określające stan związanego z nią klastra:

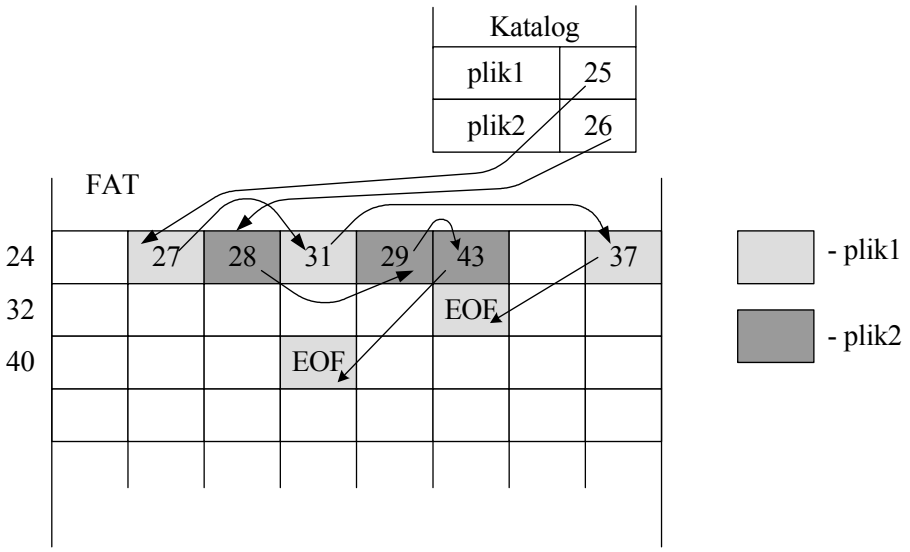
- klaster wolny (niewykorzystany),
- klaster jest zajęty przez plik i nie jest to ostatni klaster pliku (pozycja zawiera numer następnego klastra pliku),
- ostatni klaster pliku,
- klaster uszkodzony,
- klaster zarezerwowany.



Rys. 6.9. Fizyczna struktura systemu plików FAT

W stanie początkowym (po operacji formatowania) wszystkie klastry partycji są wolne, a wszystkie pozycje FAT (z wyjątkiem tych, które są związane z klastrami zarezerwowanymi) posiadają wartość „klaster wolny". Podczas umieszczania pliku na dysku system operacyjny przegląda FAT szukając pierwszej wolnej pozycji. Po znalezieniu, w pole „numer pierwszego klastra" katalogu zostaje wpisany numer tego klastra, a do klastra zostają zapisane dane pliku. W przypadku gdy plik mieści się w jednym klastrze, do danej pozycji FAT jest wpisywana informacja „ostatni klaster pliku". W przeciwnym wypadku system operacyjny poszukuje kolejnego wolnego klastra, a po jego znalezieniu do poprzedniej pozycji FAT zostaje wpisany numer tego klastra, który jest kolejnym klastrem pliku. Proces powtarza się tak długo, aż zostaną umieszczone na dysku wszystkie dane pliku. W ten sposób powstaje lista klastrów wchodzących w skład danego pliku.

W okresie początkowym po formatowaniu partycji pliki będą rozmieszczane w kolejnych (sąsiednich) klastrach obszaru danych. Jednak w wyniku usuwania plików i zapisu na ich miejsce nowych, poszczególne pliki zostają pofragmentowane i umieszczone w różnych (niesąsiadujących ze sobą) klastrach (rys. 6.10).



Rys. 6.10. Wskaźniki na pliki w FAT

Rozmiar FAT oraz liczba bitów w jednej pozycji tej tablicy jest zależna od liczby klastrów w partycji dyskowej. Z punktu widzenia maksymalnego wykorzystania powierzchni dyskowej rozmiar klastra powinien być możliwie najmniejszy (równy rozmiarowi jednego sektora – 512 B). Powoduje to jednak zwiększenie rozmiaru FAT, a także wydłuża czas trwania operacji dyskowych. Podczas formatowania dysku dla systemu FAT istnieje możliwość wyboru rozmiaru klastra, zazwyczaj z przedziału od 1 do 128 sektorów (od 512 B do 64 kB). Liczba bitów w jednej pozycji FAT musi być tak dobrana, aby zapewnić zapis maksymalnego numeru klastra dla danej partycji. W praktyce stosowane są FAT o rozmiarze pozycji 12, 16 oraz 32 bity – FAT12, FAT16, FAT32. W systemie FAT12 może wystąpić maksymalnie $2^{12} = 4096$ klastrów w obszarze danych dysku, w FAT16 – $2^{16} = 64$ k klastrów, a w systemie FAT32 – $2^{32} = 4$ G klastrów. W rzeczywistości liczba klastrów jest nieznacznie mniejsza ze względu na to, że wybrane wartości w pozycji FAT muszą określać przypadki szczególne: „Ostatni klaster pliku”, „Klaster wolny”, „Klaster uszkodzony”, „Klaster zarezerwowany”.

Z zasady formatowanie FAT12 jest stosowane dla dysków o rozmiarach nie większych od 16 MB (4 k klastrów o rozmiarze 4 kB). Analogicznie FAT16 jest wykorzystywany dla dysków o rozmiarze do 512 MB (64 k klastrów o rozmiarze 8 kB). Formatowanie FAT32 z rozmiarem klastra 4 kB jest stosowane dla dysków o pojemności do 8 GB, natomiast dla dysków większych przyjmowany jest rozmiar klastra 8, 16 lub 32 kB. Maksymalny rozmiar partycji dyskowej dla FAT16 wynosi 4 GB (64 k klastrów po 64 kB), natomiast dla FAT32 – 256 TB (4 G klastrów po 64 kB).

Podczas operacji usuwania pliku z systemu plików FAT, do pierwszego bajtu odpowiedniej pozycji katalogu zostaje wpisany znak określający, że dana pozycja jest wolna, a do wszystkich pozycji FAT związanych z tym plikiem zostaje wpisana informacja „klaster wolny". Pozostałe informacje zawarte w danej pozycji katalogu (w tym także numer pierwszego klastra pliku) pozostają niezmienione. Daje to możliwość odzyskania pomyłkowo usuniętych plików. Ze względu na wpisy do pozycji FAT związanych z usuniętym plikiem wartości „klaster wolny" i zniszczenie informacji o kolejnych klastrach zajętych przez plik, pełne automatyczne odzyskanie pliku po jego usunięciu jest możliwe jedynie w przypadku gdy plik zajmował kolejne klastry. W przeciwnym wypadku niezbędna jest analiza zawartości wolnych klastrów w celu skompletowania usuniętego pliku, co może okazać się operacją bardzo żmudną.

Podczas wszystkich operacji w systemie plików, zapisy do FAT oraz kopii FAT odbywają się jednocześnie. Dzięki temu w przypadku uszkodzenia obszaru dysku zajętego przez FAT, podczas operacji odtwarzania zawartości dysku można wykorzystać kopię FAT. Zapis informacji o rozmieszczeniu pliku na dysku w postaci listy powoduje, że system FAT jest bardzo zawodny, gdyż przerwanie listy w jednym miejscu (wadliwy jeden zapis w liście) powoduje utratę całego pliku.

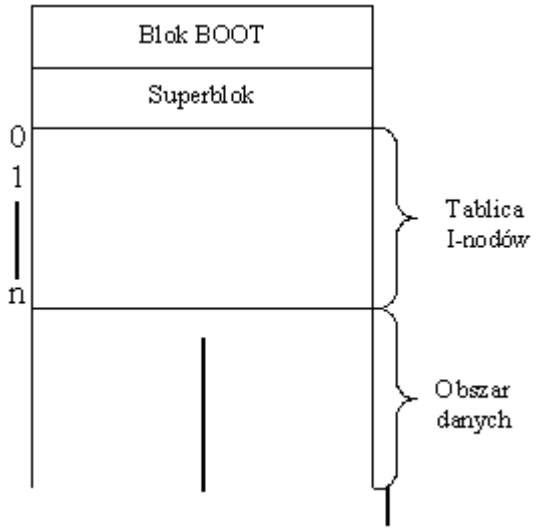
Systemy FAT12 i FAT16 stosowały 12-znakowe nazwy plików i katalogów, zgodnie z zasadą „8.3". W wersji FAT16 systemu operacyjnego Windows NT wprowadzono nowy rodzaj pozycji katalogu – „długa nazwa", co pozwoliło na stosowanie nazw złożonych z 255 znaków zapisanych w dwubajtowym kodzie Unicode. Nazwy krótkie (12-znakowe) są przechowywane analogicznie jak poprzednio, natomiast nazwy długie zostają zapisane w 26-bajtowych (13 znaków Unicode) polach umieszczonych w kolejnych pozycjach katalogu (każda pozycja zawiera 32 B). System FAT32 umożliwia także stosowanie długich nazw.

6.2.4. Fizyczna organizacja systemów S5 oraz UFS

System plików S5 został wprowadzony w UNIX System V przez Bell Labs firmy AT&T, a system UFS (Unix File System) rozszerza możliwości systemu S5 w zakresie obsługi dużych dysków oraz plików, a także zwiększa niezawodność systemu plików (zgodnie z nazewnictwem wprowadzonym w systemach UNIX, w miejsce pojęcia „klaster" będzie stosowane pojęcie „blok").

W systemie UFS partycja dyskowa zawiera cztery obszary (rys. 6.11):

- blok ładujący,
- superblok zawierający ogólne informacje o systemie plików, takie jak: rozmiar systemu plików, rozmiar obszaru zajętego przez węzły identyfikacyjne plików, liczba I-nodów, lista wolnych bloków oraz wolnych I-nodów,
- obszar węzłów identyfikacyjnych plików (I-nodów); każdy węzeł jest identyfikowany numerem,
- obszar danych, w którym są umieszczone pliki zwykłe oraz katalogi, włączając katalog główny.



Rys. 6.11. Obszary danych na dysku w systemie S5

Zasadniczą cechą systemu S5 jest oddzielenie nazwy pliku od jego opisu zawartego w specjalnej strukturze, określanej mianem węzła identyfikacyjnego pliku (ang. I-node). W systemie S5 zajmuje on 64 B i zawiera informacje o typie pliku, prawach dostępu do pliku a także o rozmieszczeniu pliku na dysku.

Do głównych pól węzła identyfikacyjnego pliku należy zaliczyć:

- identyfikator właściciela pliku,
- typ pliku (plik zwykły, katalog, plik specjalny, potok, dowiązanie symboliczne),
- prawa dostępu do pliku,
- czasy dostępu do pliku: czas ostatniej modyfikacji, czas ostatniego dostępu oraz czas ostatniej modyfikacji I-noda,
- liczba dowiązań twardych do pliku, odpowiadająca liczbie nazw, jakie posiada ten plik w hierarchii katalogów,
- tablica adresów dyskowych danych wchodzących w skład pliku,
- rozmiar pliku.

Każdy węzeł identyfikacyjny pliku posiada swój numer, który jednocześnie pełni rolę unikalnego identyfikatora pliku. Wszystkie I-nody są umieszczone w odrębnym obszarze partycji dyskowej w kolejności zgodnej z ich numerami. Należy zwrócić uwagę na fakt, że w węźle identyfikacyjnym nie jest określona nazwa pliku. Związek ten jest określony przy pomocy plików katalogowych, zorganizowanych w strukturę hierarchiczną. Plik katalogowy zawiera dwie kolumny: nazwa pliku lub podkatalogu przypisanego do danego katalogu oraz odpowiadający mu numer węzła identyfikacyjnego. Pierwsze dwie pozycje w pliku katalogowym posiadają nazwę ‘.’ (kropka) oraz ‘..’ (dwie kropki). Kropka oznacza katalog bieżący, natomiast dwie kropki katalog nadrzędny. Dla obydwu pozycji są określone numery węzłów identyfikacyjnych, opisujących

odpowiednie pliki katalogowe. Dzięki tym pozycjom system operacyjny określa hierarchię katalogów w systemie plików. Przykładowy plik katalogowy przedstawiono na rys. 6.12. Ponieważ zawartość katalogu głównego w partycji jest opisana przy pomocy odpowiedniego pliku katalogowego, więc w systemie nie jest ograniczona liczba pozycji w katalogu głównym (jak to miało miejsce w systemie FAT). System operacyjny tworzy tablicę wolnych węzłów identyfikacyjnych plików. Podczas tworzenia nowego pliku lub katalogu przydziela mu określony I-node, a podczas usuwania – zwalnia.

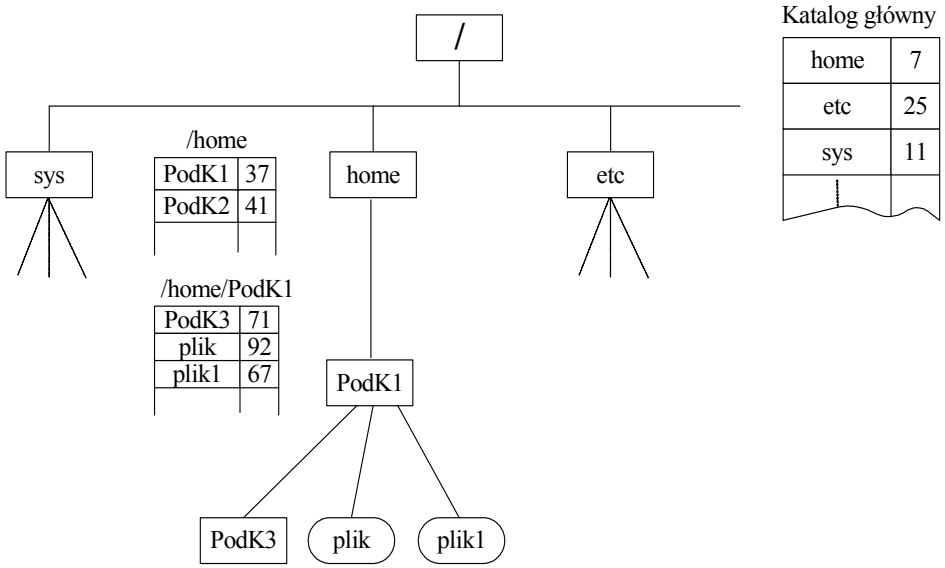
Nazwa	Numer I-noda
.	125
..	273
plik 1	103
plik 2	1271
podkatalog 1	1731
.	.
.	.
.	.

Rys. 6.12. Struktura katalogu w systemie S5

Dostęp do pliku odbywa się poprzez zastosowanie procedury kolejnego przeglądu plików katalogowych, wchodzących do ścieżki dostępu do pliku oraz odpowiadających im węzłów identyfikacyjnych. Rozpatrzmy tę procedurę na przykładzie pliku `/home/Podk1/plik`, wchodzącego w skład systemu plików przedstawionego na rys. 6.13. Określenie fizycznego adresu tego pliku (numeru węzła identyfikacyjnego) odbywa się w następujących krokach:

- przeszukiwanie katalogu głównego (ang. root) w celu określenia numeru węzła identyfikacyjnego katalogu `home` (w podanym przykładzie – 7),
- odczyt węzła identyfikacyjnego pliku o numerze 7,
- odczyt pliku katalogowego określonego przez węzeł identyfikacyjny o numerze 7,
- liniowe przeszukiwanie pliku katalogowego w celu odszukania podkatalogu `PodK1` i określenie numeru węzła identyfikacyjnego opisującego ten podkatalog (w podanym przykładzie – 37),
- odczyt węzła identyfikacyjnego o numerze 37,
- odczyt pliku katalogowego określonego przez węzeł o numerze 37,

- liniowe przeszukiwanie pliku katalogowego opisującego ten podkatalog w celu określenia numeru węzła identyfikacyjnego opisującego `plik` (w podanym przykładzie – 92),
- określenie bloków danych, w których zapisany jest plik `/home/PodK1/plik`.



Rys. 6.13. Wyznaczanie adresu pliku na podstawie jego nazwy

Opisana procedura wymaga wielu odwołań do dysku, zależnie od długości ścieżki dostępu do pliku. W celu przyspieszenia operacji plikowych, podczas operacji otwarcia pliku system operacyjny przenosi węzeł identyfikacyjny pliku z dysku do pamięci operacyjnej.

6.2.5. System plików NTFS

Podstawowe cechy systemu

W celu zwiększenia niezawodności pracy pamięci dyskowej system NTFS umożliwia odtwarzanie systemu plików w oparciu o operacje transakcyjne. Operacje transakcyjne są techniką wprowadzania modyfikacji do bazy danych tak, aby awaria systemu nie wpływała na poprawność lub integralność bazy danych. Podstawową zasadą operacji transakcyjnych jest to, że pewna grupa operacji na bazie danych, zwana transakcją, jest wykonywana w całości lub nie jest wykonywana w ogóle. W przypadku operacji dyskowych transakcja jest zdefiniowana jako operacja WE/WY, która zmienia dane w systemie plików lub zmienia strukturę katalogów danego wolumenu. Jeśli awaria systemu przerywa transakcję, wszystkie operacje wchodzące w skład transakcji muszą zostać

wycofane i stan dysku sprzed transakcji musi być przywrócony, tak jakby transakcja ta nie miała miejsca.

NTFS stosuje redundancję (nadmiarowość) do zapisu krytycznych danych systemowych, tworząc kopie zapasowe, dające możliwość odtworzenia istotnych struktur danych przy uszkodzeniu nośnika magnetycznego. Ponadto możliwa jest obsługa macierzy dyskowych poziomu pierwszego (*level 1 – mirror*), oraz poziomu piątego (*level 5 – stripes*).

NTFS stosuje 64-bitowe numery klastrów, co daje możliwość obsługi dysków o rozmiarach 2^{64} klastrów, z których każdy może mieć rozmiar do 64 kB.

Każda jednostka informacji związana z plikiem, włączając jego nazwę, właściciela, time stamps, a także jego zawartość jest implementowana jako atrybut pliku. Każdy atrybut zawiera pojedynczy strumień (ang. stream) będący po prostu sekwencją bajtów. System NTFS pozwala tworzyć nowe atrybuty, co umożliwia zwielokrotnianie strumieni danych i w konsekwencji daje wielodostęp do danych w pliku. NTFS udostępnia jeden domyślny strumień danych, nieposiadający nazwy. Aplikacja może stworzyć dodatkowy strumień związany z plikiem i nadać mu nazwę identyfikującą ten strumień. Nazwa strumienia podawana jest razem z nazwą (oddzielona dwukropkiem), np.:

```
Myfile.dat:stream2
```

Nazwy plików i katalogów zapisywane są w 16-bitowym kodzie Unicode i mogą zawierać 255 znaków, włączając spacje oraz wielokrotne kropki. NTFS spełnia standard POSIX.1 i dlatego w nazwach plików i katalogów rozróżniane są duże i małe litery oraz możliwe są tzw. dowiązania twarde (ang. hard links).

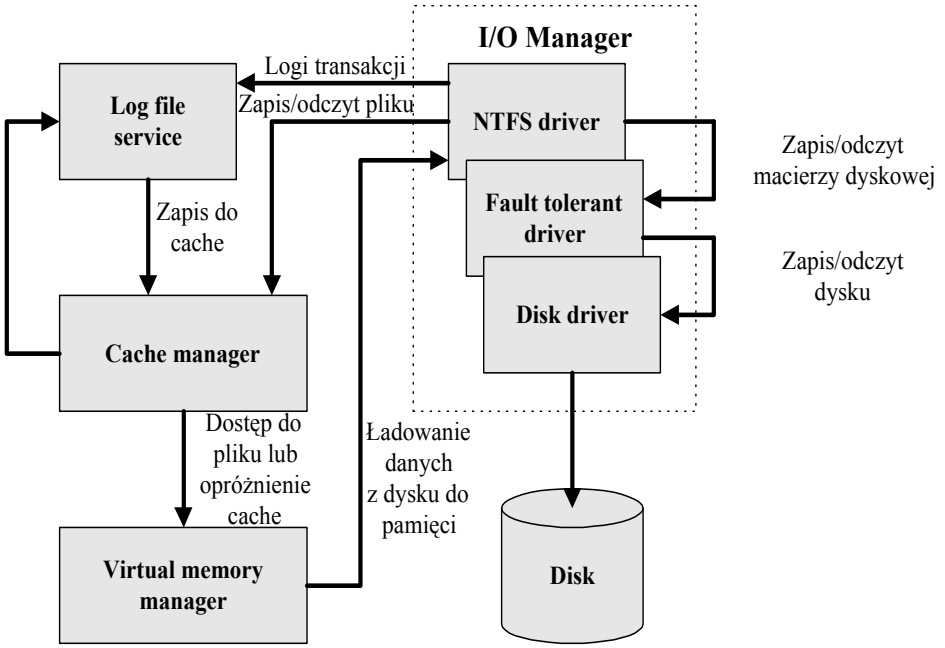
Wewnętrzna struktura systemu NTFS

Ogólną budowę komponentów programowych związanych z NTFS przedstawiono na rys. 6.14.

Obsługa pliku logów (ang. log file service – LFS) wykonuje zapisy do pliku logów, który wykorzystywany jest do odtwarzania wolumenu NTFS w przypadku uszkodzenia systemu plików. Menedżer pamięci cache (ang. cache manager) obsługuje buforowanie (ang. caching) danych dla wszystkich systemów plików, włączając także sieciowy system plików. Wykorzystywany jest do tego celu menedżer pamięci wirtualnej. Gdy program próbuje dostępu do części pliku, którego nie ma aktualnie w pamięci cache (ang. cache miss – chybienie), menedżer pamięci wywołuje NTFS i otrzymuje fragment pliku z dysku. Menedżer pamięci cache stosuje algorytm „leniwego zapisu" (ang. lazy writer).

System NTFS partycypuje w modelu obiektowym Windows poprzez implementowanie plików jako obiekty. Pozwala to na współdzielony dostęp do plików oraz na ochronę realizowaną przez menedżera obiektów (komponent Windows zarządzający wszystkimi obiektami). Aplikacja tworząca lub żądająca dostępu do pliku wykonuje tę operację analogicznie jak do każdego innego obiektu, a mianowicie z wykorzystaniem uchwytu do obiektu (ang. object

handler). Menedżer obiektów oraz system ochrony sprawdzają, czy wywołujący proces posiada prawa dostępu do pliku. System ochrony sprawdza czy znacznik dostępu (ang. access token) programu wywołującego znajduje się na liście dostępu dla obiektu.



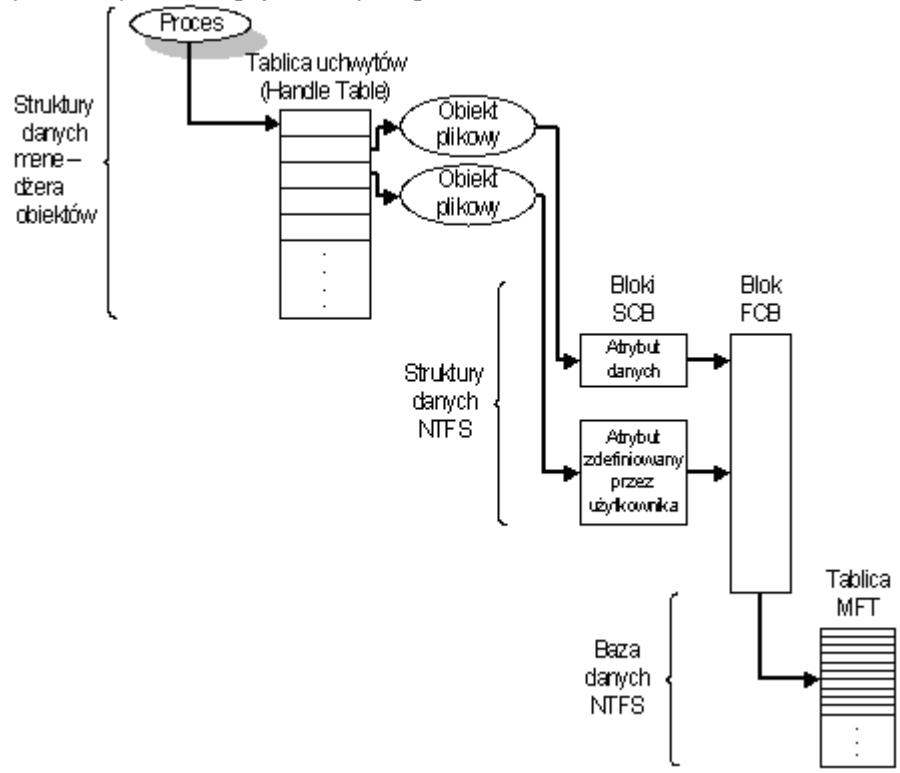
Rys. 6.14. Komponenty związane z systemem NTFS

Na rys. 6.15 przedstawiono struktury danych łączące uchwyt do pliku ze strukturą systemu plików. Na rysunku tym obiekt plikowy (ang. file object), reprezentujący pojedyncze otwarcie pliku, wskazuje na blok SCB (ang. Stream Control Block) związany z atrybutem pliku, który ma być czytany lub zapisywany. W tym przypadku proces otworzył atrybut danych oraz atrybut zdefiniowany przez użytkownika. Bloki SCB reprezentują indywidualne atrybuty plików i zawierają informacje o tym jak znaleźć konkretny atrybut w pliku. Wszystkie bloki SCB danego pliku wskazują na wspólną strukturę, określaną mianem FCB (ang. File Control Block), zawierającą wskaźnik na rekord w tablicy MFT (ang. Master File Table), umieszczoną na dysku.

Dyskowe struktury danych NTFS

Wolumen odpowiada logicznej partycji dysku i jest tworzony podczas formatowania dysku lub jego części dla NTFS. Można także stworzyć wolumen odporny na uszkodzenia, obejmujący wiele dysków. System NTFS obsługuje każdy z wolumenów osobno. Wolumen zawiera szereg plików oraz pewną dodatkową, niealokowaną przestrzeń na tej samej partycji. Wolumen NTFS

przechowuje wszystkie dane systemowe, takie jak bitmapy i katalogi, a nawet systemowy bootstrap, jako zwykłe pliki.



Rys. 6.15. Struktury danych NTFS

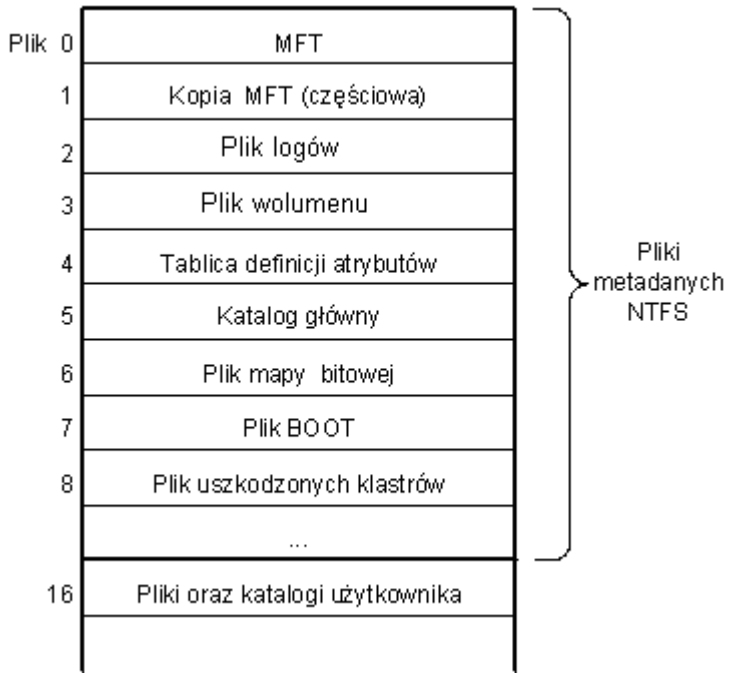
Klaster jest logiczną jednostką danych na dysku, używaną przez system plików i stanowi wielokrotność sektora, równą potędze liczby 2. Rozmiar klastra przyjmowany jest w zależności od wielkości wolumenu według zasady: im większy wolumen, tym większy rozmiar klastra.

W systemie NTFS wszystkie dane przechowywane w wolumenie dyskowym, nie wyłączając systemowych struktur danych (lokalizacja plików na dysku, bootstrap, mapy bitowe opisujące zajętość wolumenu), są przechowywane w plikach. Pozwala to swobodnie przenosić systemowe struktury danych w ramach danego wolumenu.

Najistotniejszym elementem struktury wolumenu NTFS jest *tablica MFT (ang. Master File Table)*, zaimplementowana jako tablica rekordów umieszczonych w pliku. Rozmiar każdego rekordu jest stały i wynosi 1 kB, niezależnie od rozmiaru klastra. Logicznie tablica MFT zawiera jeden wiersz dla każdego pliku w wolumenie, włączając wiersz dla samej tablicy MFT. Dodatkowo w każdym wolumenie NTFS znajduje się szereg plików metadanych, zawierających informacje używane do zaimplementowania struktury systemu plików. Każdy z tych plików posiada nazwę zaczynającą się znakiem dolara $ (np. plik zawierający tablicę MFT posiada nazwę $MFT).

Pozostałe pliki wolumenu NTFS są „normalnymi” plikami użytkownika oraz katalogami. Strukturę tablicy MFT przedstawiono na rys. 6.16.

Zwykle poszczególne rekordy MFT odpowiadają różnym plikom. Jeżeli jednak plik posiada dużą liczbę atrybutów lub jest bardzo pofragmentowany, do opisu pliku może być niezbędna większa liczba rekordów tablicy MFT. W takich przypadkach pierwszy rekord, zawierający lokalizację pozostałych, jest określany mianem bazowego rekordu pliku (ang. base file record). Przed pierwszym dostępem do wolumenu dyskowego system NTFS musi go domontować. Aby to wykonać, NTFS poszukuje pliku BOOT w celu określenia fizycznego adresu tablicy MFT, w której pozycja zerowa i pierwsza opisują sam plik MFT oraz jego częściową kopię. Częściowa kopia MFT (plik $MFTMirr) zawiera kopie szeregu początkowych wierszy tabeli MFT, opisujących pliki metadanych NTFS (rys. 6.16).



Rys. 6.16. Rekordy tablicy MFT, opisujące pliki metadanych NTFS

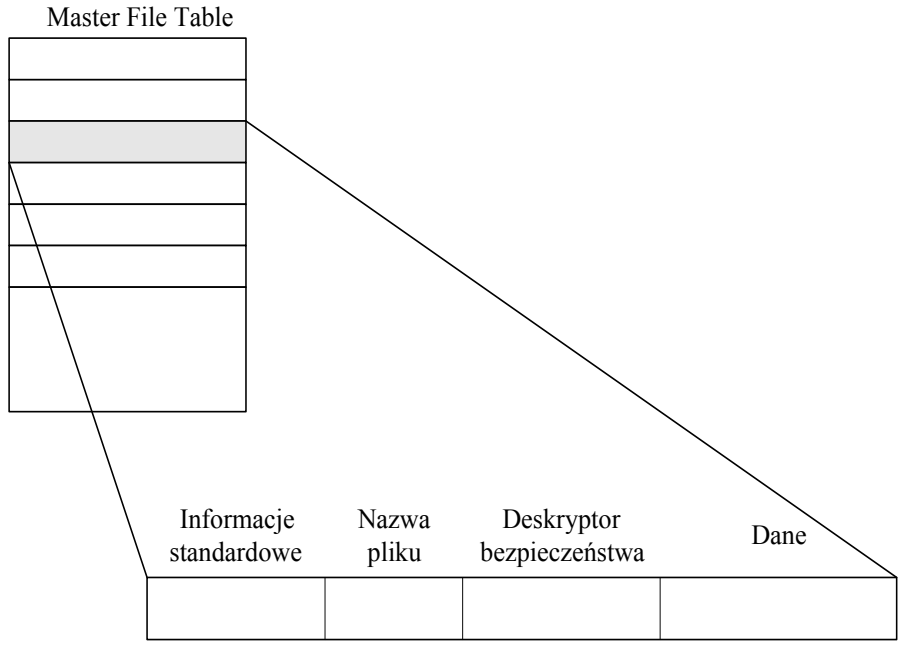
Pliki metadanych NTFS zawierają następujące dane:

- plik logów (ang. log file o nazwie $LogFile) zawiera informacje o wszystkich operacjach wykonanych w danym wolumenie NTFS. Zapisane informacje służą do odtworzenia wolumenu dyskowego po awarii,
- plik zawierający spis katalogu głównego danego wolumenu (ang. root directory),
- plik mapy bitowej (ang. bitmap file o nazwie $Bitmap), w którym każdy bit reprezentuje jeden klaster w danym wolumenie i określa czy klaster jest wolny, czy został przydzielony do jednego z plików,

- plik BOOT o nazwie $Boot zawiera program będący ogniwem podczas startu systemu operacyjnego,
- plik uszkodzonych klastrów (ang. bad-cluster file o nazwie $BadClus) zawiera informacje o uszkodzonych obszarach wolumenu,
- plik wolumenu (ang. volume file o nazwie $Volume) zawiera nazwę wolumenu, wersję NTFS, dla której wolumen był formatowany, oraz bit ustawiony w przypadku, gdy wystąpiło uszkodzenie systemu plików, i podczas kolejnego uruchomienia musi być wykonany program naprawy systemu plików,
- plik definicji atrybutów (ang. attribute definition table o nazwie $AttrDef) definiuje typy atrybutów udostępnione w wolumenie.

Pliki w wolumenie NTFS są identyfikowane przez 64-bitową liczbę, zawierającą numer pliku (ang. file number) oraz liczbę sekwencyjną (ang. reference sequence number). 48-bitowy numer pliku odpowiada numerowi pozycji tablicy MFT, opisującej dany plik, natomiast 16-bitowa liczba sekwencyjna jest każdorazowo inkrementowana podczas ponownego użycia danego rekordu MFT. Liczba ta jest wykorzystywana przez NTFS do sprawdzania wewnętrznej spójności systemu plików.

NTFS przechowuje pliki w postaci zbioru par atrybut/wartość, gdzie jednym z atrybutów są dane zawarte w pliku (ang. unnamed data attribute – nienazwany atrybut danych). Pozostałe atrybuty to: nazwa pliku, date stamp, deskryptor bezpieczeństwa, a także możliwe dodatkowe atrybuty. Na rys. 6.17 przedstawiono rekord tablicy MFT dla małego pliku.



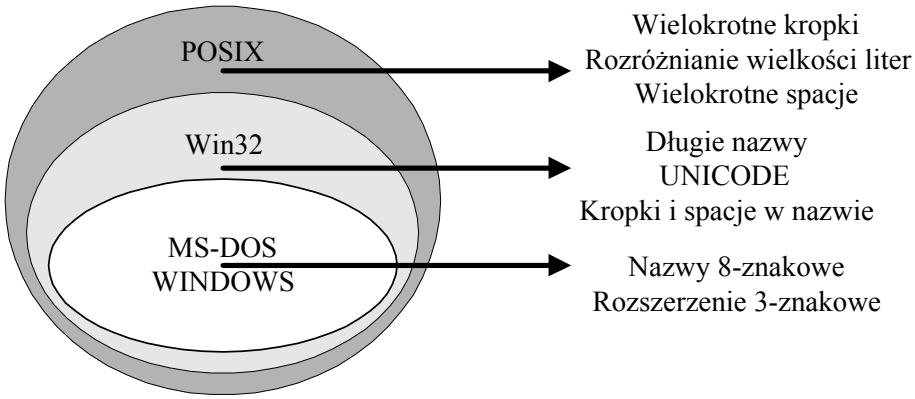
Rys. 6.17. Rekord tablicy MFT dla małego pliku

Każdy atrybut pliku jest przechowywany jako osobny strumień bajtów. Ściśle mówiąc, NTFS nie tyle odczytuje i zapisuje pliki, ile odczytuje i zapisuje strumienie atrybutów. System NTFS dostarcza następujących operacji na atrybutach: tworzenia (create), usuwania (delete), czytania (read) oraz zapisu (write). Zazwyczaj operacje odczytu i zapisu wykonywane są na nienazwanych atrybutach danych, lecz program wywołujący może określić inny atrybut. W tabeli 6.1 zestawiono standardowe atrybuty plików w wolumenie NTFS (nie wszystkie atrybuty występują dla każdego pliku).

Tabela 6.1

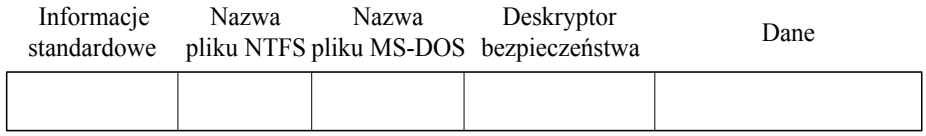
Atrybut	Opis
Informacje standardowe	Takie atrybuty, jak: read-only, archive itd.; time stamp (włączając czas utworzenia, ostatniej modyfikacji); liczba dowiązań twardych.
Nazwa pliku	Nazwa pliku w kodzie Unicode. W przypadku gdy do pliku stworzonych jest wiele dowiązań twardych oraz gdy dla pliku o długiej nazwie generowana jest krótka, 8-znakowa nazwa dla DOS, a także 16-bitowych aplikacji Windows, plik może posiadać wiele atrybutów nazw.
Deskryptor bezpieczeństwa	Struktura danych bezpieczeństwa chroniąca plik przed nieautoryzowanym dostępem; określa właściciela pliku oraz użytkowników, którzy mają dostęp do pliku.
Dane	Zawartość pliku. W NTFS plik posiada jeden domyślny nienazwany atrybut danych. Katalog nie posiada domyślnego atrybutu danych.
Indeksy	Atrybuty stosowane do określenia rozmieszczenia plików wchodzących w skład katalogu na dysku.
Lista atrybutów	Lista atrybutów tworzących plik oraz referencja do rekordu tablicy MFT, w którym każdy atrybut jest umieszczony. Ten rzadko używany atrybut jest obecny w przypadku, gdy plik wymaga więcej niż jednego rekordu MFT.

Tak NTFS, jak i FAT32 umożliwiają stosowanie nazw plików zawierających 255 znaków. Nazwy plików mogą zawierać znaki Unicode, jak również wielokrotne kropki „.” oraz spacje. Jednak FAT stosowany w DOS umożliwia nazwy 8-znakowe (w kodzie ASCII) z kropką i 3-znakowym rozszerzeniem. Na rys. 6.17 przedstawiono graficznie reprezentację przestrzeni nazw określonych przez systemy: POSIX, Win32 oraz DOS.



Rys. 6.17. Przestrzenie nazw w systemie Windows

W przypadku gdy jest tworzony plik o nazwie typowej dla Win32, NTFS automatycznie tworzy alternatywną nazwę, odpowiadającą formatowi MS-DOS, która może być traktowana jako alias. Strukturę rekordu tablicy MFT dla pliku z automatycznie wygenerowaną nazwą odpowiadającą formatowi MS-DOS przedstawiono na rys. 6.18.



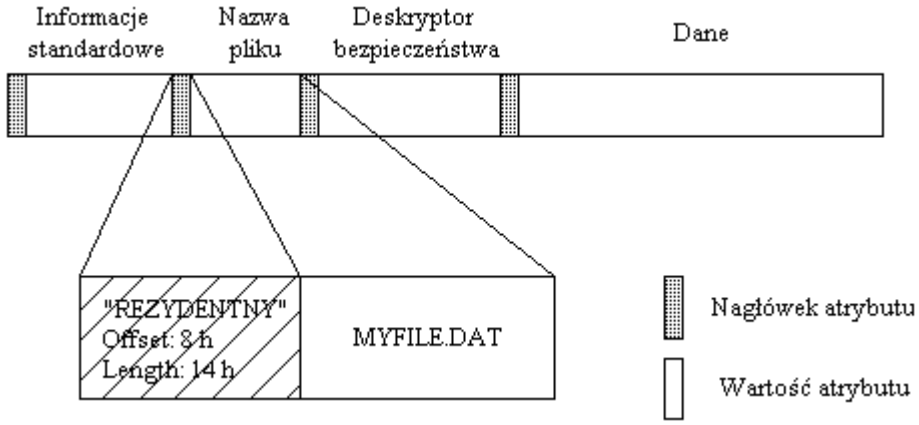
Rys. 6.18. Format rekordu tablicy MFT z atrybutem nazwy zgodnej z wymogami MS-DOS

Nazwa pliku w formacie MS-DOS może być używana do otwierania, odczytu, zapisu lub kopiowania pliku. Jeżeli zostanie zmieniona jedna z nazw pliku, to NTFS samoczynnie zmienia drugą nazwę.

W podobny sposób zaimplementowano w systemie NTFS dowiązania twarde (ang. hard links). Podczas tworzenia dowiązania twardego dla standardu POSIX system NTFS dodaje do odpowiedniego rekordu tablicy MFT nowy atrybut z nową nazwą, dzięki czemu dany plik posiada dwie lub więcej nazw. Podczas usuwania dowiązania twardego usuwany jest atrybut nazwy z rekordu tablicy MFT, a gdy atrybut nazwy jest jeden, to usuwany jest plik.

W przypadku małych plików wszystkie atrybuty i ich wartości są zapisane w rekordzie tablicy MFT (rekord tablicy MFT posiada rozmiar 1 kB). Jeżeli wartość atrybutu mieści się w rekordzie MFT, to atrybut określony jest mianem atrybutu rezydentnego. Każdy atrybut zaczyna się standardowym nagłówkiem, który NTFS wykorzystuje do zarządzania atrybutami. Nagłówek, który jest zawsze rezydentny, zawiera informację o tym czy wartość atrybutu jest rezydentna, czy nie. Dla atrybutów rezydentnych nagłówek zawiera także informację o adresie (offsecie) wartości atrybutu względem nagłówka oraz

długość wartości atrybutu. Na rys. 6.19 przedstawiono rezydentny atrybut nazwy pliku.



Rys. 6.19. Nagłówek i wartość rezydentnego atrybutu

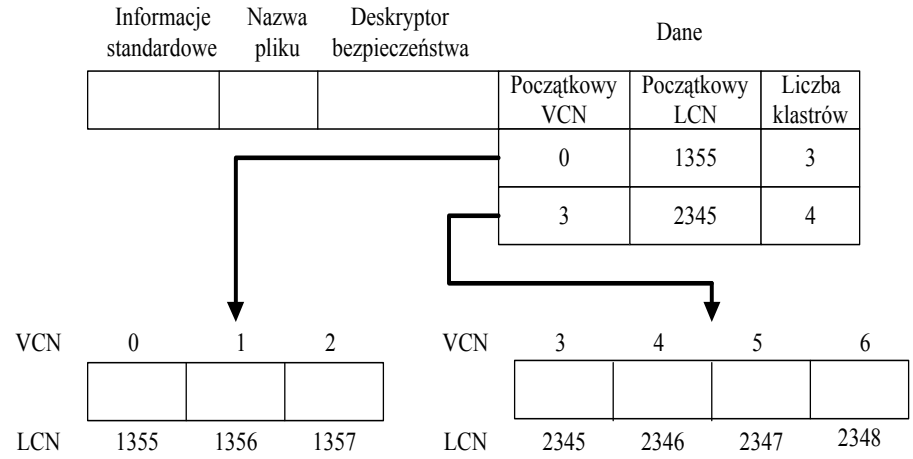
W przypadku gdy wartości wszystkich atrybutów pliku są zapisane w rekordzie tablicy MFT, dostęp do pliku jest najszybszy. Jeżeli jednak plik nie mieści się w rekordzie tablicy MFT (rozmiar 1kB), system musi poszukiwać opisu pliku w tej tablicy, a następnie odczytywać poszczególne jednostki alokacji z dysku. Wydłuża to czas dostępu do plików o większych rozmiarach.

Jeśli któryś z atrybutów (najczęściej atrybut danych) nie mieści się w rekordzie tablicy MFT, system NTFS alokuje klastry danego wolumenu (spoza tablicy MFT) do jego zapisu, określane w języku angielskim mianem *extents*. Atrybuty, których wartości przechowywane są w ekstentach, określane są mianem *atrybutów nierezydentnych*. W takim przypadku nagłówek atrybutu, znajdujący się w rekordzie tablicy MFT, zawiera informację o rozmieszczeniu poszczególnych ekstentów w wolumenie.

Każdy klaster w wolumenie posiada swój numer, określany mianem LCN (ang. Logical Cluster Number), poprzez który system NTFS identyfikuje daną jednostkę alokacji. Do określania kolejności ekstentów w ramach danego pliku stosowany jest numer ekstentu o nazwie VCN (Virtual Cluster Number). System NTFS zapisuje w nagłówku danego atrybutu kolejne ekstenty, zawierające wartości danego atrybutu, poprzez podanie:

- początkowego numeru VCN,
- początkowego numeru LCN,
- liczby kolejnych klastrów.

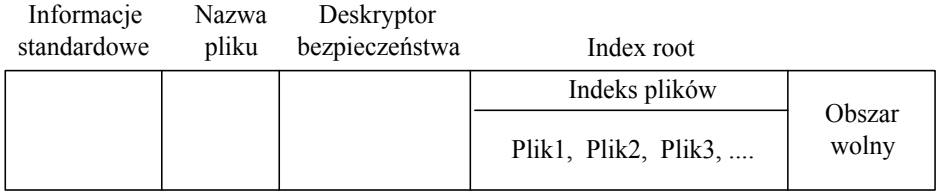
W ten sposób w nagłówku atrybutu nie są zapisywane wszystkie numery klastrów zawierających wartość atrybutu, lecz jedynie początkowy numer klastra oraz liczba klastrów. Daje to dużą oszczędność miejsca, gdyż jeżeli wartość atrybutu zapisana jest w wielu kolejnych klastrach, to w nagłówku atrybutu odpowiada temu jedna pozycja. Przedstawiono to na rys. 6.20.



Rys. 6.20. Translacja VCN na LCN dla nierezydentnego atrybutu danych

Dla bardzo dużych plików oraz plików bardzo pofragmentowanych może okazać się, że w rekordzie tablicy MFT nie można pomieścić informacji o wszystkich ekstentach, w których zapisany jest plik. W takim przypadku do opisu jednego pliku wykorzystywana jest większa liczba rekordów tablicy MFT. Pierwszy rekord, określony mianem rekordu bazowego, zawiera lokalizację pozostałych rekordów tablicy MFT.

Mały katalog mieści się w jednym rekordzie tablicy. W takim przypadku atrybut o nazwie pień indeksu (ang. index root) zawiera indeks referencji do plików oraz podkatalogów w tablicy MFT, przypisanych do danego katalogu (rys. 6.21).

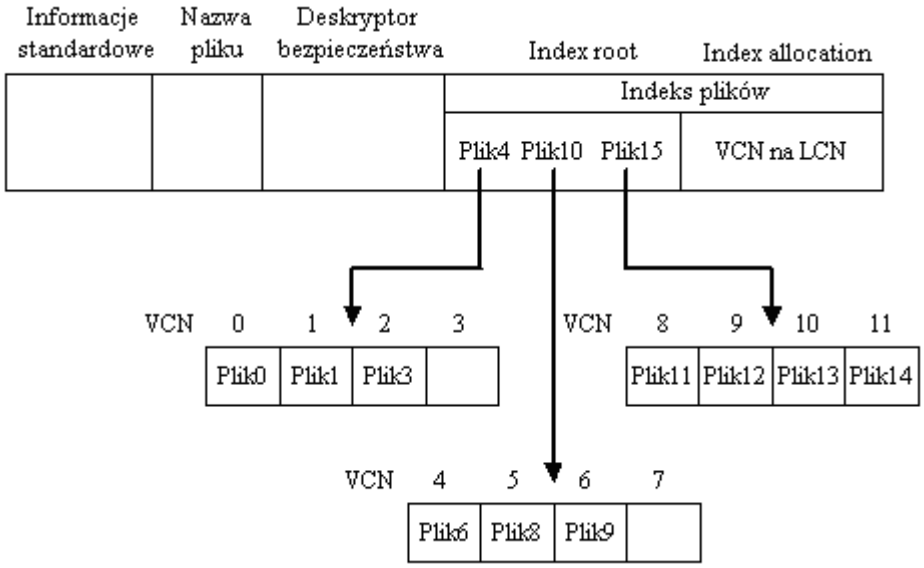


Rys. 6.21. Rekord tablicy MFT dla małego katalogu

Indeks jest kolekcją nazw plików wraz z numerami referencyjnymi, zorganizowaną tak, aby zminimalizować czas dostępu do plików. W celu stworzenia katalogu system NTFS indeksuje atrybuty nazw plików. Rekord tablicy MFT dla katalogu głównego danego wolumenu przedstawiono na rys. 6.22.

Pozycja tablicy MFT dla katalogu zawiera w atrybucie index root posortowaną listę plików przypisanych do danego katalogu. Jednak dla dużych katalogów nazwy plików są zapisane poza rekordem tablicy, w tzw. buforach indeksowych o rozmiarze klastra. Bufory indeksowe implementują w postaci drzewa binarnego (B+tree) strukturę danych, która minimalizuje liczbę

dostępów do dysku, niezbędnych do odszukania potrzebnego pliku. W atrybucie indeks główny rekordu tablicy MFT zapisany jest pierwszy poziom drzewa binarnego, a jego pozycje wskazują na bufory indeksowe, zawierające niższy poziom drzewa binarnego. Z kolei atrybut indeksowej alokacji (ang. index allocation), służy do odwzorowania kolejnych numerów klastrów (VCN) na logiczne numery jednostek alokacji w wolumenie (LCN), określające miejsce na dysku, w którym zapisano dany bufor indeksowy. W jednym buforze może pomieścić się 20 do 30 pozycji plików lub podkatalogów.



Rys. 6.22. Budowa pliku katalogowego dla dużego katalogu

W rekordzie tablicy MFT, opisującym długi katalog, atrybut indeks główny zawiera szereg nazw plików pełniących rolę indeksu do niższego poziomu drzewa binarnego. Każdy plik w atrybucie indeks główny posiada (opcjonalny) związany z nim wskaźnik, wskazujący na odpowiedni bufor indeksowy. Bufor indeksowy natomiast zawiera uporządkowaną alfabetycznie listę plików, których nazwy są „wcześniej w alfabecie", niż nazwa pliku wskazującego na dany bufor indeksowy.

6.3. Operacje plikowe

System operacyjny udostępnia użytkownikom szereg operacji na plikach, w postaci wywołań funkcji systemowych. Niezależnie od tego jakie operacje na plikach będą wykonywane, system operacyjny musi wykonywać uniwersalne operacje związane z opisem plików:

- określenie charakterystyki pliku na podstawie jego nazwy symbolicznej,
- skopiowanie charakterystyki pliku do pamięci operacyjnej,

- określenie praw użytkownika do wykonania określonej operacji (np. odczyt, zapis, usunięcie itp.) na podstawie opisu pliku,
- zwolnienie pamięci operacyjnej po charakterystyce pliku.

Ponadto każda operacja plikowa zawiera szereg działań, jak np. odczyt oraz zapis określonego zbioru klastrów, usunięcie pliku itp.

6.3.1. Otwarcie pliku

W systemie UNIX wywołanie systemowe `open` wykorzystuje dwa parametry: nazwę pliku wraz ze ścieżką dostępu oraz tryb otwarcia pliku. Tryb otwarcia wskazuje systemowi operacyjnemu jakie operacje zamierzamy wykonywać na pliku aż do momentu jego zamknięcia (np. wyłącznie operacje odczytu, wyłącznie zapisu czy odczytu i zapisu). Podczas otwarcia pliku system operacyjny wyznacza przede wszystkim numer węzła identyfikacyjnego pliku na podstawie ścieżki dostępu do pliku, a następnie odczytuje I-noda z dysku i kopiuje go do pamięci operacyjnej. Informacje zawarte w węźle identyfikacyjnym pliku zostają następnie przez system operacyjny uzupełnione o następujące pola:

- stan I-noda w pamięci, określający:
  - czy plik jest zablokowany,
  - czy jakiś proces oczekuje na zdjęcie blokady z pliku,
  - czy kopia I-noda w pamięci jest identyczna z oryginałem na dysku,
  - czy zawartość pliku w pamięci różni się od oryginału na dysku,
- logiczny numer urządzenia dyskowego zawierającego plik,
- numer I-noda (w tablicy I-nodów na dysku numer I-noda nie jest zapisany, gdyż jest określony przez pozycję w tej tablicy),
- licznik odwołań do I-noda.

Jeden plik w danej chwili może być jednocześnie wykorzystywany przez wiele procesów. Mimo to system operacyjny przechowuje w pamięci jedną kopię I-noda, określaną mianem V-noda. Podczas kolejnego otwarcia pliku system operacyjny sprawdza, czy w pamięci jest umieszczony I-nod tego pliku, a jeżeli tak, to licznik odwołań do I-noda jest zwiększany o 1. Analogicznie, podczas zamknięcia pliku licznik odwołań do I-noda zostaje zmniejszony o 1, a jeżeli przyjmuje wartość 0, to bufor przechowujący I-noda zostaje zwolniony.

Poza I-nodem przechowującym ogólne informacje o pliku, system operacyjny tworzy strukturę zawierającą informacje niezbędne dla konkretnego procesu korzystającego z danego pliku. Struktura ta (typu `file`) jest przechowywana w pamięci operacyjnej w tablicy otwartych plików. Podczas otwarcia pliku przez jeden z procesów system operacyjny sprawdza prawa dostępu procesu do pliku i jeżeli operacja ta przebiegła pomyślnie, to w tablicy

otwartych plików tworzona jest nowa pozycja, zawierająca strukturę typu `file`. Struktura `file` zawiera następujące pola:

- tryb otwarcia pliku (wyłącznie do odczytu, do odczytu i zapisu itp.),
- wskaźnik na strukturę I-noda,
- bieżącą pozycję w pliku (offset) dla operacji odczytu/zapisu,
- licznik odwołań do I-noda,
- wskaźnik na strukturę zawierającą prawa procesu otwierającego plik (struktura ta znajduje się w deskryptorze procesu),
- wskaźnik na poprzednią i następną strukturę `file` (2-kierunkowa lista struktur `file`).

Zmienna `offset` w strukturze `file` pozwala na zapamiętanie bieżącego położenia w pliku. Podczas otwarcia pliku zmienna ta zawiera początkową lub końcową (w zależności od trybu otwarcia) pozycję w pliku. Operacje odczytu oraz zapisu danych powodują zmianę bieżącej pozycji w pliku. Program użytkowy może zmieniać aktualne położenie w pliku przy pomocy funkcji systemowej `lseek`.

Przy każdym otwarciu jakiegokolwiek pliku system operacyjny tworzy nową strukturę `file`, umieszcza ją w tablicy otwartych plików, a w tablicy deskryptorów procesu umieszcza wskaźnik na tę strukturę (rys. 6.23). Gdy proces otwiera dany plik wielokrotnie, dla każdego otwarcia system operacyjny tworzy nową strukturę `file`. Proces potomny, który dziedziczy kontekst procesu macierzystego (a w tym także tablicę deskryptorów), uzyskuje możliwość wykonywania operacji na plikach otwartych w procesie macierzystym.

6.3.2. Wymiana danych z plikiem

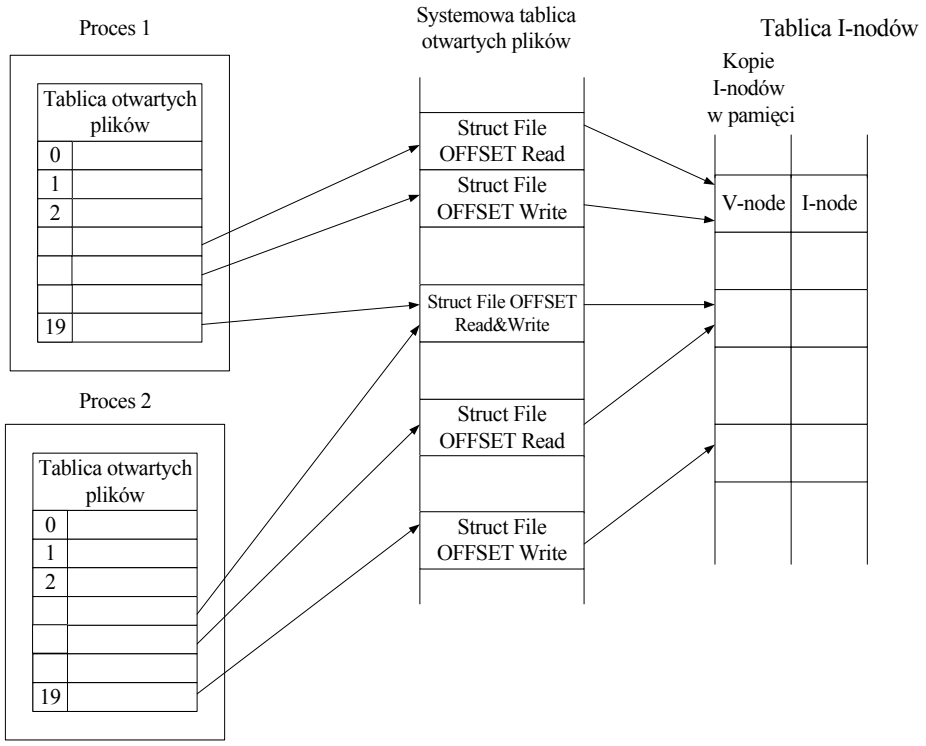
Do wymiany danych z otwartym dyskiem w systemie operacyjnym UNIX przewidziano funkcje `read` oraz `write`. Funkcja systemowa `read` wykorzystuje trzy parametry:

```
Read(deskr, bufor, l_bajtów);
```

Parametr `deskr` jest liczbą całkowitą określającą numer pozycji w tablicy deskryptorów procesu, przydzielonej danemu plikowi. Drugim parametrem jest wskaźnik na tablicę znaków, w której zostanie umieszczony odczytany fragment pliku. Liczbę odczytywanych znaków określa się przy pomocy trzeciego parametru `l_bajtów`. Funkcja `read` zwraca rzeczywistą liczbę odczytanych bajtów (liczba ta może się różnić od parametru `l_bajtów`) lub wartość -1 oznaczającą błąd odczytu. Odczyt danych z pliku rozpoczyna się od aktualnej pozycji wskaźnika w pliku (`offset`) przechowywanej w strukturze `file`. Po wykonaniu operacji odczytu `offset` zostaje zwiększony o liczbę przeczytanych bajtów.

Funkcja `write` wykonuje zapis do pliku określonego przez zmienną `deskr`, liczbę bajtów `l_bajtów` z bufora określonego przez wskaźnik `bufor`:

```
write(deskr, bufor, l_bajtów);
```



Rys. 6.23. Związki procesów z otwartymi plikami

Funkcja zwraca liczbę rzeczywiście zapisanych bajtów lub kod błędu -1.

Obydwie operacje służące do wymiany danych z plikiem są operacjami synchronicznymi, co oznacza, że podczas realizacji operacji odczytu/zapisu proces zostaje zablokowany aż do jej zakończenia.

6.3.3. Blokady plików

Blokady plików oraz poszczególnych rekordów w plikach służą do synchronizacji procesów korzystających ze wspólnych plików. Wielozadaniowe systemy operacyjne z zasady obsługują specjalne wywołanie systemowe pozwalające programiście na ustawianie i sprawdzanie blokad na plikach oraz ich częściach. W systemie UNIX służy do tego funkcja `fcntl`. Parametrami tej funkcji są: deskryptor pliku, typ operacji (ustawienie lub sprawdzenie blokady,

blokowanie dostępu dla odczytu lub zapisu), a także obszar blokowania – `offset` oraz liczba bajtów.

Podczas wywołania funkcji `fcntl` w celu sprawdzenia blokad pliku, funkcja natychmiast zwraca do procesu informację o nałożonych blokadach. Natomiast nałożenie blokady może odbywać się dwojako:

- z przejściem procesu w stan oczekiwania, w przypadku gdy nałożenie blokady jest niemożliwe (tzw. synchroniczne wywołanie systemowe),
- z natychmiastowym powrotem do procesu wywołującego oraz zwróceniem kodu błędu (tzw. asynchroniczne wywołanie systemowe).

Blokada pliku do zapisu nie może być zrealizowana w przypadku, gdy inny proces wcześniej ustawił blokadę do zapisu na tym pliku. Oznacza to, że blokada pliku do zapisu jest blokadą wyłączną. Blokady pliku do odczytu nie są wyłączne, co oznacza, że wiele procesów jednocześnie może wykonywać operacje odczytu z danego pliku. Jeżeli natomiast na jakąkolwiek część pliku nałożono blokadę odczytu, to nie można na tę część pliku nałożyć blokady zapisu.

W systemie UNIX stosowane są dwa tryby funkcjonowania blokad:

- konsultatywny (ang. advisory), w którym system operacyjny nie zajmuje się blokowaniem operacji na pliku, a jedynie ustawia wskaźniki blokady w strukturze `file`. Współbieżne procesy wykorzystujące wspólny plik muszą w takim przypadku sprawdzać występowanie blokad nałożonych na plik,
- obowiązujący (ang. mandatory), w którym system operacyjny blokuje dostęp do pliku, na który zostały nałożone blokady przez inny proces.

Podstawowym trybem pracy jest tryb konsultatywny, gdyż w niewielkim stopniu obciąża system operacyjny.

6.4. Kontrola dostępu do pliku

6.4.1. Organizacja kontroli dostępu w systemie UNIX

W systemie operacyjnym UNIX prawa dostępu do pliku lub katalogu są określone dla trzech rodzajów użytkowników:

- właściciela pliku (identyfikator UID – ang. User Identifier),
- członków grupy, do której należy właściciel (GID – ang. Group Identifier),
- pozostałych użytkowników systemu.

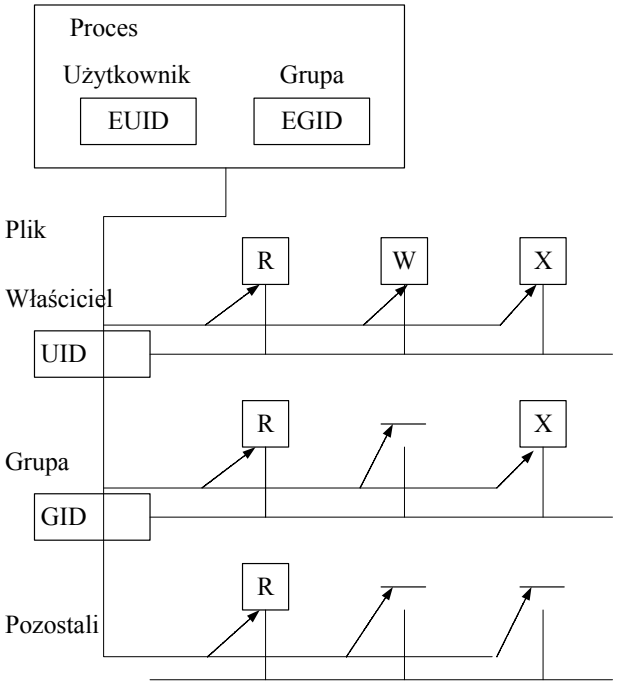
Biorąc pod uwagę, że w systemie występują trzy rodzaje uprawnień: r (odczyt), w (zapis) oraz x (wykonanie), uprawnienia dostępu do pliku (katalogu) zostają zapisane na 9 bitach.

Z każdym procesem UNIX są związane dwa identyfikatory: użytkownika, dla którego został utworzony dany proces, oraz grupy, do której należy dany użytkownik. Identyfikatory te noszą następujące nazwy: RUID (ang. Real UID) – rzeczywisty identyfikator użytkownika, oraz RGID (ang. Real GID) –

rzeczywisty identyfikator grupy użytkownika. Podczas sprawdzania uprawnień dostępu do pliku nie są jednak brane pod uwagę RUID oraz RGID, lecz tzw. uprawnienia efektywne – EUID oraz EGID. W momencie tworzenia procesu uprawnienia efektywne są identyczne jak uprawnienia rzeczywiste (rys. 6.24).

Podczas otwarcia pliku przez proces system operacyjny porównuje EUID oraz EGID z UID oraz GID otwartego pliku. Określenie uprawnień procesu do wykonywania operacji na pliku następuje na podstawie następującego algorytmu:

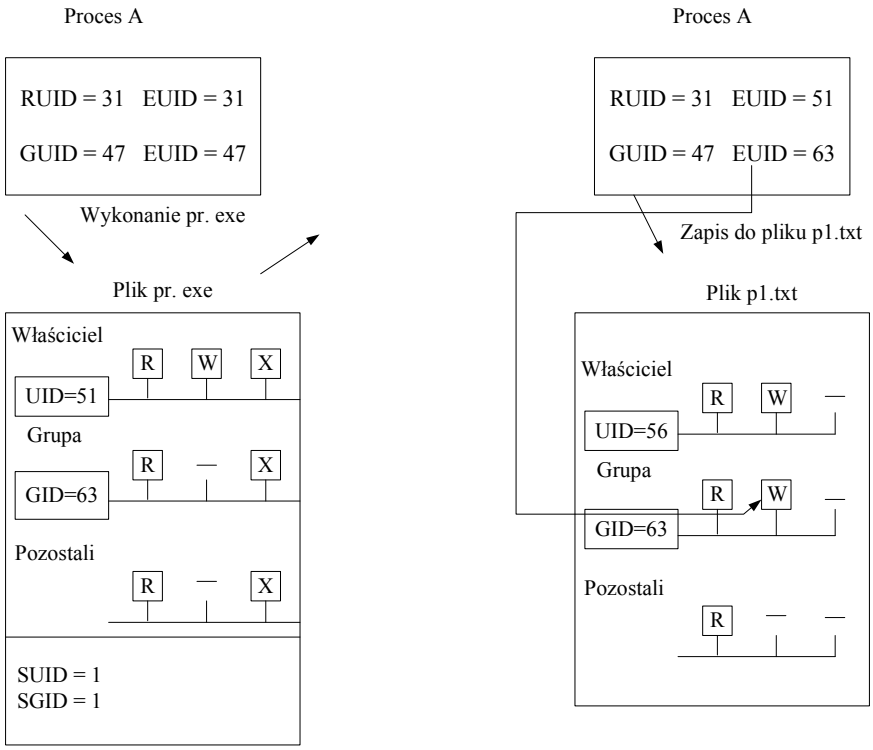
- jeżeli EUID = UID to proces posiada uprawnienia właściciela pliku,
- jeżeli EGID = GID to proces posiada uprawnienia grupy właściciela pliku,
- jeżeli EUID ≠ UID i EGID ≠ GID to proces posiada uprawnienia dla pozostałych użytkowników.



Rys. 6.24. Kontrola praw dostępu do pliku w systemie UNIX

Jednak w sytuacji, gdy dany proces wykonuje wywołanie systemowe `exec`, w procesie rozpoczyna się realizacja nowego programu pobranego z określonego pliku. Jeśli dla nowo wczytanego pliku są ustawione dwa znaczniki SUID (ang. Set UID) oraz SGID (ang. Set GID) (bity te są uzupełnieniem 9 bitów określających uprawnienia dostępu do pliku), pozwalające na zmianę identyfikatorów użytkownika i grupy podczas wykonywania tego programu, to uprawnienia efektywne EUID oraz EGID zostają zmienione na uprawnienia nowo wczytanego pliku. Omówioną sytuację

przedstawiono na rys. 6.25. Na początkowym etapie proces A posiada RUID równy EUID oraz RGID równy EGID (odpowiednio 31 i 47). Następnie w procesie wykonano funkcję `exec` powodującą załadowanie nowego programu `pr.exe`, dla którego UID = 51 i GID = 63. Ponieważ dla pliku `pr.exe` ustawione są znaczniki SUID = 1 oraz SGUID = 1, więc efektywne identyfikatory procesu A przyjmują wartości odpowiednio EUID = 51 oraz EGID = 63. Pozwala to na wykonanie operacji zapisu do pliku `p1.txt`.



Rys. 6.25. Zmiana efektywnych identyfikatorów

6.4.2. Organizacja kontroli dostępu w systemie Windows NT

Kontrola uprawnień dostępu do obiektów dowolnego typu w systemie Windows odbywa się za pomocą monitora bezpieczeństwa (ang. Security Reference Monitor), wykonywanego w trybie uprzywilejowanym. Cechą charakterystyczną systemu bezpieczeństwa jest znaczna liczba wbudowanych (predefiniowanych) podmiotów dostępu – tak poszczególnych użytkowników (Administrator, System, Guest), jak i grup (Users, Administrators, Account Operators, Server Operators, Everyone itd.). Poszczególni użytkownicy oraz grupy posiadają przydzielone określone uprawnienia, co w istotny sposób ułatwia pracę administratorowi. Podczas dodawania nowego użytkownika administrator musi jedynie podjąć decyzję, do której grupy (lub grup) go

dołączyć. Oczywiście administrator może tworzyć nowe grupy użytkowników, a także dodawać nowe uprawnienia do istniejących grup, jednak w bardzo wielu przypadkach istniejące grupy są w pełni wystarczające.

Windows NT wspiera następujące trzy klasy operacji dostępu, różniące się typem podmiotów oraz obiektów biorących w nich udział:

- zezwolenia (ang. permissions) – zbiór operacji, które mogą być określone dla wszystkich podmiotów w stosunku do obiektu dowolnego typu (plików, katalogów, drukarek, sekcji pamięci itp.). Zezwolenia odpowiadają prawom dostępu do plików i katalogów w systemie UNIX,
- prawa (ang. user rights) – są określane dla grupy w celu umożliwienia wykonywania niektórych operacji systemowych, takich jak ustawianie zegara systemowego, archiwizacja plików, wyłączenie komputera itp. W takich operacjach występuje szczególny obiekt dostępu, jakim jest cały system operacyjny. To prawa, a nie zezwolenia odróżniają jedną grupę użytkowników od drugiej,
- możliwości użytkowników (ang. user abilities) są określane dla poszczególnych osób korzystających z programu i umożliwiają formowanie środowiska pracy, jak np.: zmiana zawartości menu głównego programów, możliwość korzystania z punktu Menu Run itp.

Prawa oraz zezwolenia przydzielone danej grupie są nadane wszystkim członkom tej grupy, co pozwala administratorowi traktować ich jednakowo. Podczas przyłączania się do systemu użytkownik otrzymuje tzw. żeton dostępu (ang. access token), zawierający identyfikator użytkownika oraz identyfikatory wszystkich grup, do których należy, listę zarządzania dostępem (ang. Access Control List – ACL), zawierającą zbiór zezwoleń dla danego użytkownika, a także zbiór praw na wykonywanie określonych operacji systemowych.

W systemie NTFS administrator ma możliwość zarządzania dostępem użytkowników do poszczególnych katalogów oraz plików. Zezwolenia w Windows NT można zaliczyć do jednej z dwu grup: zezwoleń indywidualnych oraz standardowych. Zezwolenia indywidualne odnoszą się do operacji elementarnych na katalogach i plikach, natomiast zezwolenia standardowe stanowią zbiory odpowiednich zezwoleń indywidualnych. W tabeli 6.2 przedstawiono sześć zezwoleń indywidualnych (operacji elementarnych) mających znaczenie dla plików i katalogów.

Tabela 6.2

ZEZWOLENIE	DLA KATALOGU	DLA PLIKU
Read (R)	Odczyt nazw plików i katalogów wchodzących w skład danego katalogu, a także atrybutów oraz właściciela katalogu.	Odczyt danych, atrybutów, właściciela pliku.

Write (W)	Dodawanie plików i podkatalogów do katalogu, zmiana atrybutów katalogu, odczyt właściciela.	Odczyt właściciela, zmiana atrybutów pliku, zmiana oraz dopisanie danych do pliku.
Execute (X)	Odczyt atrybutów katalogu, dokonywanie zmian w podkatalogach wchodzących do danego katalogu, odczyt właściciela.	Odczyt atrybutów pliku, właściciela. Wykonanie pliku, jeśli zawiera kod programu.
Delete (D)	Usunięcie katalogu.	Usunięcie pliku.
Change Permission (P)	Zmiana zezwoleń dla katalogu.	Zmiana zezwoleń dla pliku.
Take Ownership (O)	Zmiana właściciela katalogu.	Zmiana właściciela pliku.

Dla plików przewidziano cztery zezwolenia standardowe: No Access, Read, Change oraz Full Control, grupujące następujące zezwolenia indywidualne (tabela 6.3).

Tabela 6.3

ZEZWOLENIA STANDARDOWE	ZEZWOLENIA INDYWIDUALNE
No Access	-
Read	R, X
Change	R, W, X, D
Full Control	R, W, X, D, P, O

Zezwolenie Full Control różni się od zezwolenia Change tym, że daje możliwość zmiany zezwoleń (P) oraz zmiany właściciela (O).
Dla katalogów w Windows NT określono siedem zezwoleń standardowych: No Access, List, Read, Add, Add&Read, Change oraz Full Control. W tablicy 6.4 zestawiono zbiory zezwoleń indywidualnych przypisanych poszczególnym zezwoleniom standardowym w odniesieniu do katalogów, a także informację o tym, w jaki sposób zezwolenia standardowe są przekształcane na zezwolenia dla poszczególnych plików wchodzących w skład katalogu, w przypadku gdy pliki dziedziczą zezwolenia dla katalogów.

Tabela 6.4.

ZEZWOLENIA STANDARDOWE	ZEZWOLENIA INDYWIDUALNE DLA KATALOGÓW	ZEZWOLENIA INDYWIDUALNE DLA PLIKÓW PRZY DZIEDZICZENIU
No Access	-	-
List	R, X	-

Read	R, X	R, X
Add	W, X	-
Add&Read	R, W, X	R, X
Change	R, W, X, D	R, W, X, D
Full Control	Wszystkie	Wszystkie

W przypadku gdy w katalogu ustawiony jest znacznik dziedziczenia zezwoleń, podczas tworzenia plik otrzymuje zezwolenia od katalogu, w którym został stworzony.

6.5. Funkcjonowanie dyskowej pamięci cache

W wielu systemach operacyjnych odwołania do blokowych urządzeń zewnętrznych (a w tym przede wszystkim do pamięci dyskowych) są przechwytywane przez podsystem buforowania, stanowiący pośrednią warstwę programową, określaną także mianem pamięci cache. Oprogramowanie to znajduje się między warstwą systemu plików a sterownikiem dyskowym.

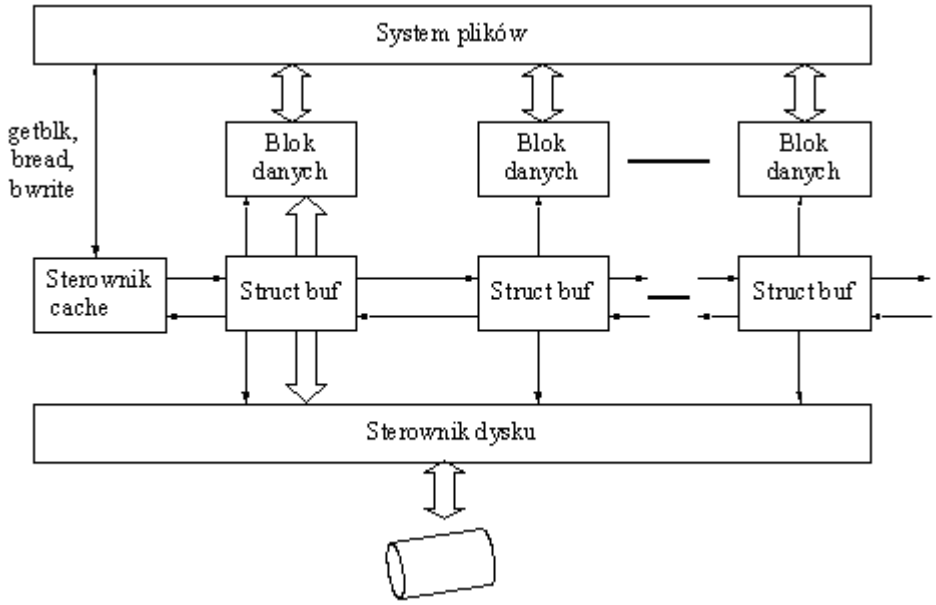
W wyniku operacji odczytu określonego bloku danych z pamięci dyskowej sterownik cache przegląda swój katalog i jeżeli potrzebny blok znajduje się w pamięci cache, to sterownik kopiuje z niej potrzebne dane do pamięci procesu. W wyniku tego zostaje zrealizowana operacja WE/WY, mimo że nie była realizowana operacja dostępu do pliku. Podczas zapisu danych do pliku, dane zapisywane są także do bufora pamięci cache, a właściwy zapis danych do pliku odbywa się podczas usuwania bloku danych z tej pamięci (pamięć cache z odroczonym zapisem – ang. Write-Back) lub na żądanie programu użytkowego. W wyniku tego dane w pamięci cache mogą być wielokrotnie modyfikowane, nim zostaną fizycznie zapisane na dysku. W wielu systemach operacyjnych dyskowa pamięć cache zajmuje znaczną część pamięci operacyjnej. Ma to szczególnie miejsce w serwerach plików.

Wadą dyskowej pamięci cache jest potencjalne obniżenie niezawodności systemu. Podczas załamania się systemu, gdy z różnych przyczyn (zanik zasilania, błąd kodu systemu operacyjnego itp.) tracimy dane przechowywane w pamięci operacyjnej, mogą zostać także stracone dane, które wcześniej zostały zapisane do pliku, jednak ze względu na funkcjonowanie dyskowej pamięci cache nie zostały jeszcze fizycznie zapisane na dysku. Aby uniknąć takiej sytuacji system operacyjny powinien cyklicznie opróżniać bufory dyskowe i fizycznie zapisywać dane na dysku (przykładem może być wywołanie systemowe `sync` w systemie UNIX).

Stosowane są dwie metody organizacji dyskowej pamięci cache. Pierwszy z nich, który można określić jako tradycyjny, jest oparty na autonomicznym sterowniku tej pamięci, który samodzielnie realizuje wszystkie operacje związane z funkcjonowaniem buforów dyskowych. W taki sposób były organizowane dyskowe pamięci cache w wielu systemach UNIX, a także NetWare. Drugi sposób organizacji tej pamięci jest oparty na wykorzystaniu możliwości podsystemu pamięci wirtualnej.

6.5.1. Tradycyjna dyskowa pamięć cache

Każde odwołanie do operacji dyskowej zostaje skierowane do podsystemu buforowania dysku, który składa się ze zbioru buforów dyskowych oraz szeregu programów określanych mianem sterownika dyskowej pamięci cache. Rozmiar poszczególnych buforów jest równy rozmiarowi bloku (klastra) w systemie operacyjnym (logiczna jednostka danych). Z każdym buforem jest związany nagłówek (struktura `buf`), zawierający następujące pola: adres bufora w pamięci operacyjnej, typ operacji (odczyt/zapis), adres bloku na dysku (numer dysku, numer partycji dyskowej, logiczny numer bloku). Wszystkie nagłówki buforów zostają połączone w listę dwukierunkową (rys. 6.26).

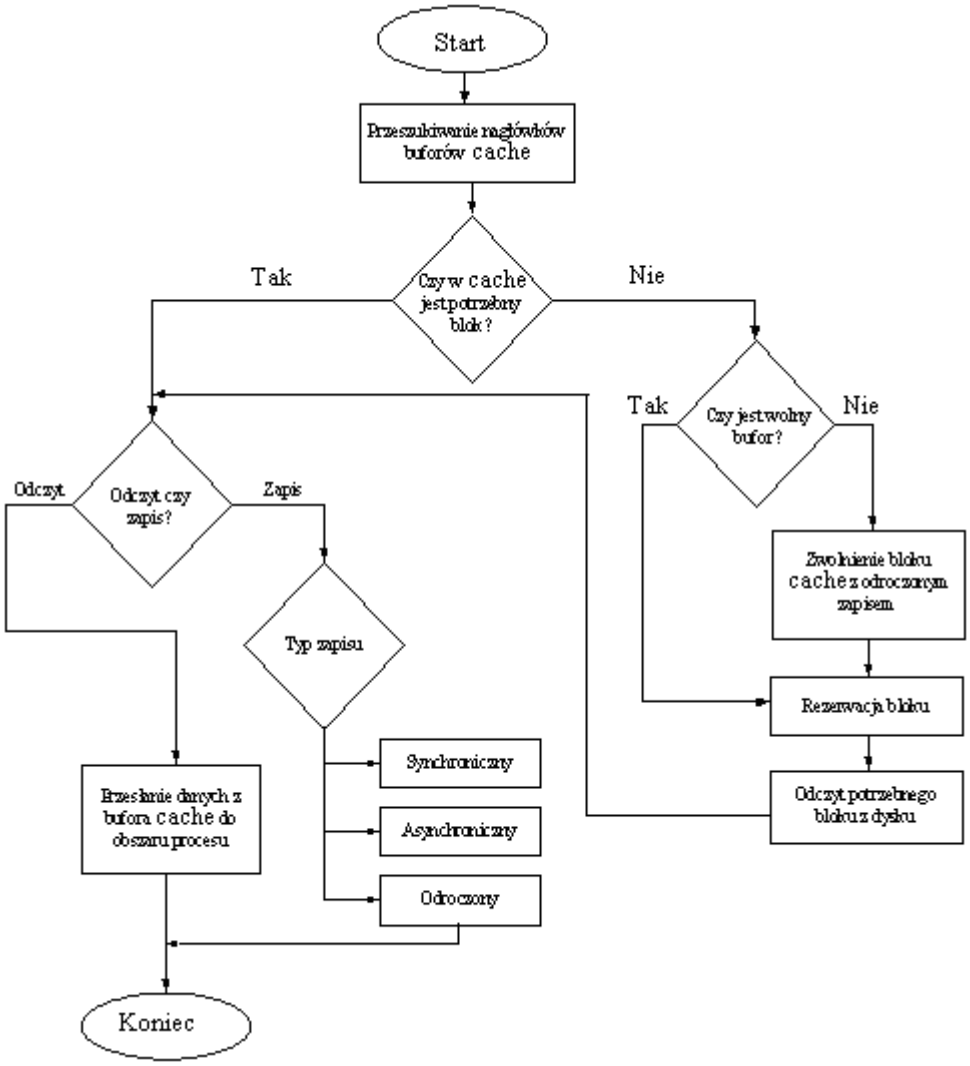


Rys. 6.26. Organizacja tradycyjnej dyskowej pamięci cache

Interfejs dyskowego sterownika cache wykonuje następujące operacje:

- zapis synchroniczny. Podczas tej operacji następuje fizyczny zapis danych z bufora na dysk, podczas gdy proces, który wykonał tę operację, przechodzi w stan oczekiwania,
- zapis asynchroniczny. Analogicznie jak poprzednio wykonywany jest fizyczny zapis danych z bufora na dysk, jednak proces, który wykonał tę operację, nie jest wstrzymywany,
- zapis odroczony. Zapis danych nie jest wykonywany, jednak do nagłówka zostaje wpisana informacja o tym, że bufor może być przepisany na dysk jeśli istnieje potrzeba jego opróżnienia,
- odczyt danych z bufora. Poszukiwanie bufora zawierającego określony blok danych. Jeśli w pamięci cache takiego bloku nie ma, to poszukiwany jest wolny bufor lub opróżniany bufor z zapisem odroczonym i następuje odczyt zadanego bloku danych z dysku.

Uproszczony algorytm funkcjonowania dyskowej pamięci cache przedstawiono na rys. 6.27.



Rys. 6.27. Uproszczony algorytm funkcjonowania dyskowej pamięci cache

6.6. Stabilność systemów plików

W rozdziale poddano analizie metody, które zwiększają stabilność systemu komputerowego podczas awarii dyskowych.

6.6.1. Rekonstrukcja systemu plików

Możliwość rekonstrukcji systemu plików to właściwość, która gwarantuje, że w przypadku zaniku zasilania lub załamania systemu, gdy wszystkie dane przechowywane w pamięci operacyjnej zostają bezpowrotnie stracone, rozpoczęte operacje plikowe mogą być szybko zakończone lub wycofane bez ujemnego wpływu dla zdolności pracy systemu plików.

Dowolna operacja na pliku (tworzenie, usunięcie, zapis, odczyt itd.) może być przedstawiona w postaci określonej sekwencji podoperacji. Skutki zaniku zasilania lub załamania systemu operacyjnego zależą od tego, jaka operacja WE/WY była wykonywana w tym momencie, w jakiej kolejności były realizowane poszczególne podoperacje oraz jaka podoperacja została wykonana jako ostatnia. Rozpatrzmy dla przykładu skutki zaniku napięcia zasilania podczas operacji usuwania pliku w systemie FAT. W celu wykonania tej operacji należy usunąć zapis w odpowiednim pliku katalogowym, a także wyzerować wszystkie elementy FAT, które odpowiadały klastrom usuwanego pliku. Załóżmy, że zanik napięcia zasilania miał miejsce po usunięciu zapisu w pliku katalogowym i wyzerowaniu części elementów FAT (lecz nie wszystkich). W takim przypadku system plików może pracować nadal, jednak ostatnie klastry usuwanego pliku zostają na zawsze stracone, gdyż pozostają zaznaczone jako „zajęte". Sytuacja byłaby znacznie gorsza, gdyby usuwanie pliku rozpoczynało się od wyzerowania klastrów zajętych przez plik. W takim przypadku zanik napięcia zasilania między tymi operacjami spowodowałby, że zawartość pliku katalogowego nie odpowiadałaby rzeczywistemu stanowi systemu plików (plik istnieje, a w rzeczywistości go nie ma). Nieusunięta pozycja katalogu zawiera numer pierwszego klastra nieistniejącego pliku, a ponieważ ten klaster został zaznaczony jako wolny, więc może być przydzielony innemu plikowi. Może to być źródłem wielu konfliktów.

Uszkodzenie systemu plików może także wynikać z wadliwego funkcjonowania dyskowej pamięci cache. Podczas buforowania danych występują bardzo często sytuacje, gdy zawartość bufora dyskowego różni się od aktualnej zawartości pliku. Dotyczy to nie tylko plików użytkownika, lecz także plików wykorzystywanych przez system operacyjny, takich jak pliki katalogowe, węzły identyfikacyjne plików itp. Zanik napięcia w takiej sytuacji może doprowadzić do bardzo poważnej awarii systemu plików.

W celu zapewnienia zgodności zawartości dyskowej pamięci cache z zawartością plików dyskowych, w systemie operacyjnym należy zapewnić możliwie częste przepisywanie zmodyfikowanych buforów dyskowych na dysk. Może się to odbywać na żądanie sterownika cache lub programu użytkowego. Sterownik dyskowej pamięci przepisuje bloki pamięci z cache na dysk w następujących przypadkach:

- konieczność zwolnienia miejsca w pamięci cache dla nowych danych,
- sterownik cache odebrał polecenie zapisu danych na dysk od programu użytkowego lub innego modułu systemu operacyjnego,
- cykliczny zrzut danych z wszystkich zmodyfikowanych buforów dyskowych na dysk.

Mimo że okres cyklicznego zrzutu zmodyfikowanych buforów dyskowych jest z zasady niewielki (np. od 10 do 30 s), istnieje duże prawdopodobieństwo, że przy zaniku napięcia zasilania nastąpi poważne uszkodzenie systemu plików. Z tego powodu w wielu systemach operacyjnych udostępnia się specjalne oprogramowanie pozwalające na naprawę uszkodzonego systemu plików, jak np. `fsck` w systemach UNIX, `ScanDisk` dla FAT lub `Chkdsk` dla systemu HPFS. Mimo to mogą wystąpić sytuacje, gdy naprawa systemu plików po zaniku napięcia zasilania będzie niemożliwa.

6.6.2. Przetwarzanie transakcyjne

Pod pojęciem transakcji rozumiemy zbiór operacji modyfikujących dane, które są traktowane jako niepodzielna całość i nie mogą być realizowane częściowo. Transakcja jest wykonywana w całości lub nie jest wykonywana w ogóle. W systemie plików takimi transakcjami mogą być operacje WE/WY, zmiana zawartości plików, katalogów lub innych struktur systemu plików (np. węzłów identyfikacyjnych plików w UFS czy elementów FAT).

Rozpatrzmy dowolną operację w systemie plików. Operacja ta składa się z szeregu kroków (podoperacji) związanych z tworzeniem, usuwaniem i modyfikacją obiektów wchodzących w skład systemu plików. Jeśli wszystkie podoperacje zostały zakończone z powodzeniem, to transakcję uznaje się za zatwierdzoną (ang. commit). Jeżeli natomiast jedna lub więcej podoperacji nie zakończyło się powodzeniem ze względu na zanik napięcia zasilania lub załamanie się systemu operacyjnego, to w celu zapewnienia spójności systemu plików wszystkie modyfikacje danych muszą być wycofane aż do momentu rozpoczęcia transakcji.

W systemie plików istnieje także wiele operacji niepociągających za sobą modyfikacji danych w systemie plików i dlatego nie należy ich rozpatrywać jako transakcje, np. odczyt pliku, poszukiwanie pliku na dysku, odczyt atrybutów pliku.

W celu realizacji operacji transakcyjnych system operacyjny musi zapamiętywać wszystkie operacje realizowane w ramach danej transakcji po to, aby po przerwaniu jej wycofać niezakończoną transakcję (ang. rollup). W systemach plików z dyskową pamięcią cache, w celu odtworzenia systemu po zaniku zasilania, poza wycofaniem niezakończonych transakcji należy także powtórnie wykonać zakończone transakcje, gdyż jedynie w ten sposób można odtworzyć stan buforów dyskowych.

Dla zapewnienia możliwości odtworzenia systemu plików po zaniku zasilania system operacyjny stosuje wyprzedzające protokołowanie transakcji. Polega ono na tym, że przed zmianą jakiegokolwiek bloku danych w buforze lub na dysku wykonywany jest zapis do pliku dziennika transakcji (ang. logfile) o wykonanej transakcji oraz poprzedniej i nowej zawartości bloku. Informacja o zatwierdzeniu transakcji zostaje także zapisana w pliku dziennika, a nowa zawartość bloku jest jeszcze przez pewien czas przechowywana w pliku dziennika w celu umożliwienia powtórzenia wykonanej transakcji.

6.6.3. Rekonstrukcja systemu NTFS

System plików NTFS pozwala jedynie na pełną rekonstrukcję danych systemowych, czyli: katalogów, atrybutów bezpieczeństwa, mapy bitowej zajętych klastrów oraz pozostałych plików systemowych. System nie gwarantuje pełnej ochrony danych zapisanych w plikach użytkownika. Dla ochrony danych systemowych system NTFS stosuje przetwarzanie transakcyjne, dające pewność rekonstrukcji danych systemowych. Jednak dane użytkownika, które podczas załamania systemu znajdowały się w buforach dyskowej pamięci CACHE, nie mogą być rekonstruowane.

Dziennik rejestracji transakcji w NTFS jest podzielony na dwie części - obszar restartu oraz obszar protokołowania:

- Obszar restartu zawiera informacje o tym, z którego miejsca należy odczytywać dziennik transakcji w celu wykonania procedury rekonstrukcji systemu po zaniku napięcia zasilania lub załamaniu systemu operacyjnego. Jest to wskaźnik na określony rekord w obszarze protokołowania (dla pewności w pliku dziennika tworzone są dwie kopie obszaru restartu).
- Obszar protokołowania zawiera rekordy o wszystkich zmianach w danych systemowych systemu plików, mających miejsce w wystarczająco długim okresie czasu. Każdy rekord jest identyfikowany kolejnym numerem (ang. Logical Sequence Number). Rekordy dotyczące podoperacji wykonywanej w ramach danej transakcji tworzą listę (każdy następny rekord zawiera numer rekordu poprzedniego). Obszar protokołowania posiada określony rozmiar. Oznacza to, że po zapełnieniu tego obszaru wpisy wykonywane są od początku, w miejscu starych wpisów.

Można wyróżnić kilka rodzajów rekordów występujących w dzienniku transakcji, dotyczących modyfikacji, punktu kontrolnego, zatwierdzenia transakcji, tablicy modyfikacji, tablicy zmodyfikowanych stron.

Wszystkie operacje na pliku dziennika transakcji (plik logów) są wykonywane wyłącznie przez usługę systemową LFS (ang. Log File Service). Przed wykonaniem każdej transakcji NTFS wywołuje usługę LFS w celu zarejestrowania wszystkich podoperacji, jakie będą wykonane w celu realizacji transakcji. W następnej kolejności system wykonuje wszystkie podoperacje na kopiach bloków danych, znajdujących się w buforach dyskowej pamięci cache. Po wykonaniu wszystkich podoperacji usługa LFS wpisuje do pliku dziennika transakcji rekord zatwierdzenia transakcji.

Równolegle z operacjami rejestracji i wykonania transakcji jest realizowany proces przenoszenia bloków pamięci cache na dysk. Proces ten jest wykonywany w dwóch etapach: w pierwszej kolejności przepisywane są bloki pliku dziennika, a następnie bloki danych zmodyfikowane przez transakcję. Każdorazowo gdy sterownik cache podejmuje decyzję o przeniesieniu

określonych bloków danych na dysk, informuje o tym usługę LFS. W wyniku tego LFS odwołuje się do sterownika cache w celu zapisu na dysk wszystkich zmienionych bloków dziennika transakcji. Taka dwufazowa procedura przepisywania danych z dyskowej pamięci cache na dysk zapewnia możliwość rekonstrukcji systemu w wypadku, gdy podczas zapisu na dysk zmodyfikowanych buforów pamięci cache wystąpi zanik zasilania.

6.6.4. Macierze dyskowe

Jedną z metod zwiększenia niezawodności systemu plików jest nadmiarowy zapis danych na dysku (dyskach). Rozwiązanie takie nosi nazwę macierzy dyskowej RAID (ang. Redundant Array of Inexpensive/Independent Disks). Technologia RAID polega na zapisie danych na zbiorze (macierzy) dysków w taki sposób, aby po uszkodzeniu któregokolwiek z dysków istniała możliwość odtworzenia danych.

Macierz RAID może być utworzona na bazie szeregu klasycznych dysków, zarządzanych standardowymi kontrolerami dyskowymi. W takim przypadku w systemie operacyjnym musi funkcjonować specjalny sterownik nadrzędny, sterujący zbiorem dysków. W systemie Windows NT takim sterownikiem jest FtDisk. Bardzo często są także stosowane profesjonalne macierze dyskowe, w których zbiór dysków jest sterowany specjalizowanym sterownikiem całej macierzy.

Dla użytkowników oraz programów użytkowych macierz dyskowa RAID jest jednym dyskiem logicznym o właściwościach zależnych od algorytmów zastosowanych do sterowania macierzą oraz od rozmieszczenia danych na poszczególnych dyskach macierzy. Rozwiązania stosowane w macierzach dyskowych są klasyfikowane według tzw. poziomów RAID: RAID-0, RAID-1, RAID-2, RAID-3, RAID-4, RAID-5 oraz innych, będących ich kombinacjami.

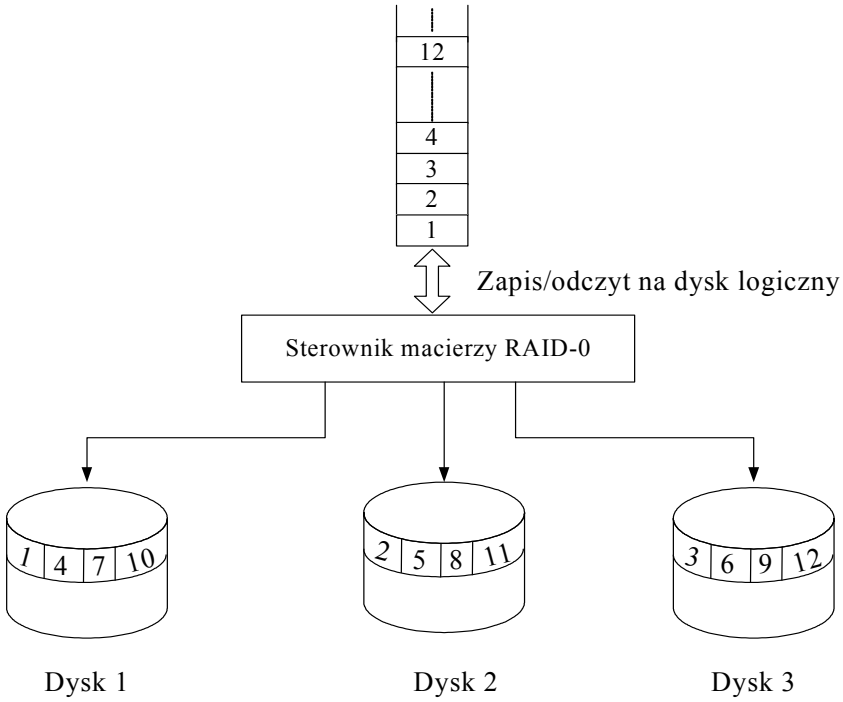
Podczas oceny efektywności macierzy RAID brane są pod uwagę przede wszystkim następujące kryteria:

- stopień nadmiarowości w przechowywaniu informacji,
- wydajność operacji odczytu i zapisu,
- stopień odporności na uszkodzenia.

W logicznym urządzeniu RAID-0 (rys. 6.28) sterownik macierzy dyskowej rozdziela zapisywane dane na bloki o określonym rozmiarze i przekazuje je równolegle na wszystkie dyski macierzy, przy czym pierwszy blok danych jest zapisywany na pierwszym dysku, drugi – na drugim itd. W Windows NT rozmiar takiego bloku wynosi 64 kB. Podczas odczytu danych sterownik macierzy multipleksuje (łączy) bloki danych odczytane z poszczególnych dysków i przekazuje do pamięci operacyjnej.

W porównaniu z pojedynczym dyskiem, w którym poszczególne bloki danych są zapisywane/odczytywane sekwencyjnie, w przypadku macierzy dyskowej RAID-0 znacznie rośnie wydajność pamięci masowej ze względu na równoległość pracy poszczególnych dysków. Macierz dyskowa RAID-0

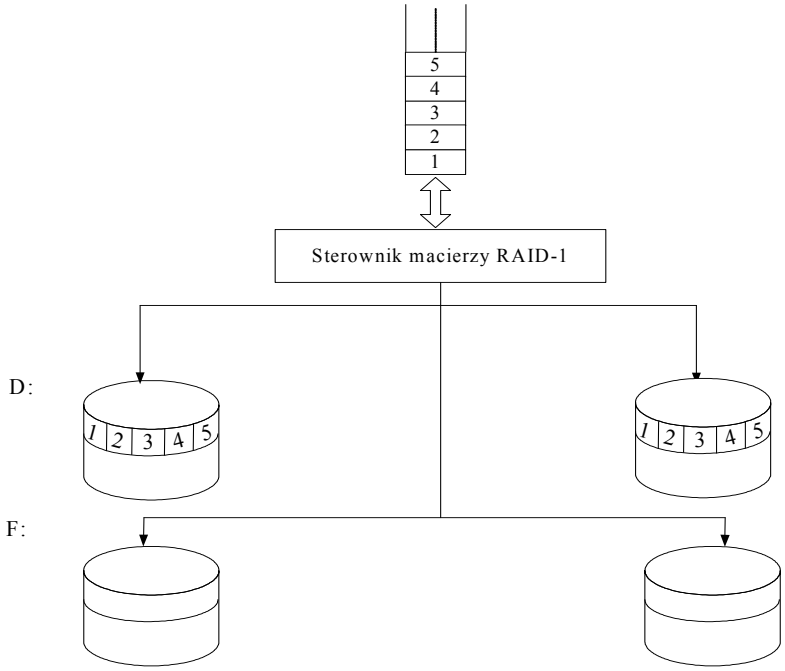
charakteryzuje się brakiem redundancji (nadmiarowości). Oznacza to, że stopień odporności macierzy dyskowej na uszkodzenia nie ulega poprawie. Można nawet powiedzieć, że odporność macierzy na uszkodzenia pogarsza się w stosunku do pojedynczego dysku, gdyż rośnie prawdopodobieństwo uszkodzenia jednego z wielu dysków wchodzących w skład macierzy, a uszkodzenie jednego z dysków powoduje awarię całej macierzy. Można wskazać jeszcze jedną wadę macierzy dyskowej RAID-0. A mianowicie, podczas konieczności uzupełnienia macierzy o jeszcze jeden dysk istnieje konieczność ponownego rozmieszczenia wszystkich bloków danych na poszczególnych dyskach macierzy.



Rys. 6.28. Organizacja macierzy RAID-0

Poziom RAID-1 (rys. 6.29) realizuje podejście określone mianem kopiowania zwierciadlanego (ang. mirroring). Urządzenie logiczne RAID-1 jest tworzone na podstawie jednej (lub wielu) pary dysków, z których jeden jest dyskiem podstawowym, a drugi – kopią zwierciadlaną. W przypadku uszkodzenia jednego z dysków, drugi zawiera nieuszkodzone dane. Macierz RAID-1 charakteryzuje się znaczną redundancją (jedynie 50% całkowitej objętości macierzy jest wykorzystana do zapisu właściwych danych). Daje to wysoką odporność macierzy RAID-1 na uszkodzenia. Podczas operacji zapisu danych sterownik macierzy zapisuje informacje jednocześnie na obydwa dyski. W przypadku gdy obydwa dyski macierzy posiadają własne sterowniki,

operacja zapisu danych jest wykonywana równolegle. Powoduje to, że operacje zapisu odbywają się z taką samą szybkością jak dla pojedynczego dysku. Operacja odczytu danych z macierzy RAID-1 może być wykonywana z dowolnego dysku. W niektórych przypadkach (np. sterowniki SCSI) istnieje możliwość jednoczesnego odczytu różnych danych z obydwu dysków macierzy, co może zwiększyć wydajność macierzy (nawet dwukrotnie).



Rys. 6.29. Organizacja macierzy RAID-1

Poziom RAID-2 charakteryzuje się bitowym rozdziałem danych między poszczególne dyski macierzy. W rezultacie, pierwszy bit danych zapisany zostaje na pierwszym dysku, drugi – na drugim itd. Dzięki równoległym operacjom odczytu/zapisu na poszczególnych dyskach uzyskuje się znaczne zwiększenie szybkości pracy pamięci masowej (macierze dyskowe zawierają od 16 do 32 dysków). Odporność na uszkodzenia uzyskuje się poprzez zastosowanie dysków nadmiarowych, przeznaczonych do zapisu kodu korekcji błędów (w przypadku liczby dysków w macierzy od 16 do 32, liczba nadmiarowych dysków wynosi 3). RAID-2 zapewnia bardzo dużą wydajność pamięci masowej, a także wysoką niezawodność. Ze względu jednak na pokaźną cenę rozwiązania takie są stosowane wyłącznie w komputerach klasy mainfraim oraz superkomputerach.

W macierzy dyskowej RAID-3 zapisywane dane zostają podzielone na poszczególne bajty, które następnie są zapisane na N-1 dyskach wchodzących w skład macierzy. Dysk o numerze N jest przeznaczony do zapisu sumy

modulo 2 (XOR), wyznaczonej dla bajtów zapisanych na N-1 dyskach danych. W przypadku gdy jeden z dysków macierzy ulega uszkodzeniu, dane zapisane na pozostałych dyskach oraz suma modulo 2 zapisana na dysku nadmiarowym pozwalają na odtworzenie uszkodzonych danych. Odtworzenie danych może odbywać się dynamicznie podczas operacji odczytu i zapisu, jak również w wyniku uruchomienia specjalnej procedury odtwarzania danych na nowo wymienionym dysku.

Rozpatrzmy np. procedurę odtwarzania danych w macierzy dyskowej zawierającej trzy dyski, z których dwa (D1, D2) zawierają dane, a dysk D3 zawiera sumę modulo 2 (XOR). Załóżmy, że do pamięci masowej zapisywane są kolejne liczby naturalne (D1 – 0000 0001, D2 – 0000 0010, D1 – 0000 0011 itd.). Zawartość poszczególnych dysków wchodzących w skład macierzy przedstawiono na rys. 6.30.

Dysk D1	Dysk D2	Dysk D3
0000 0001	0000 0010	0000 0011
0000 0011	0000 0100	0000 0111
0000 0101	0000 0110	0000 0011
0000 0111	0000 1000	0000 1111

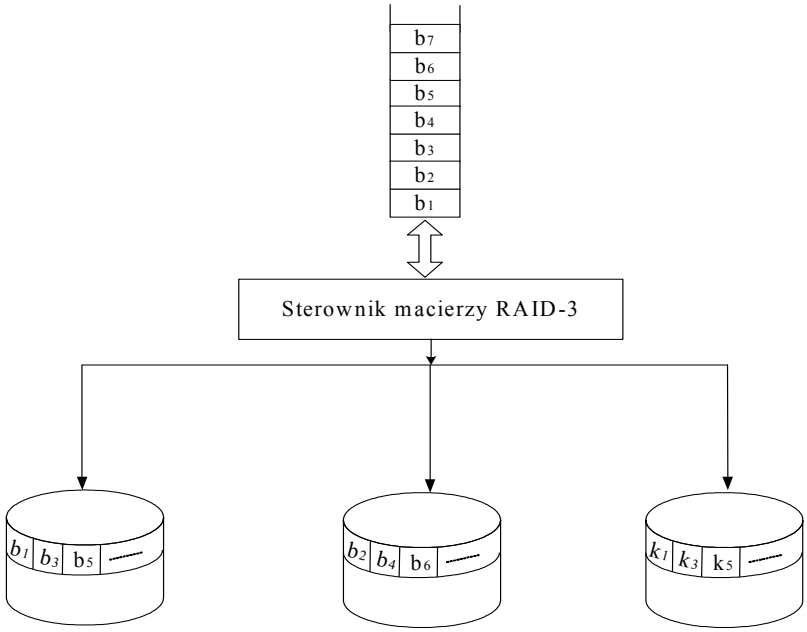
Rys. 6.30. Przykład rozmieszczenia danych na poszczególnych dyskach macierzy RAID-3

Załóżmy, że zostały uszkodzone dane 0000 0100 (drugi bajt na dysku D2). Odtworzenie tych danych jest możliwe poprzez odczyt danych zapisanych w drugim wierszu na wszystkich pozostałych dyskach macierzy oraz wyznaczenie sumy modulo 2:

$$\begin{array}{r} 0000\ 0011 \\ 0000\ 0111 \\ \hline 0000\ 0100 \end{array}$$

Macierz dyskowa RAID-3 pozwala na równoległe wykonywanie operacji zapisu/odczytu na wielu dyskach macierzy, jednak w danej chwili może być obsługiwana tylko jedna operacja WE/WY. Oznacza to, że w danej chwili może być obsługiwany wyłącznie jeden proces. Strukturę macierzy dyskowej RAID-3 przedstawiono na rys. 6.31.

Organizacja macierzy RAID-4 jest analogiczna do RAID-3 z tą różnicą, że dane zostają podzielone na bloki, a nie na bajty (jak to miało miejsce w RAID-3). Podobnie jak w RAID-3, do przechowywania danych kontrolnych używany jest jeden dodatkowy dysk. Dzięki temu, że poszczególne bloki wchodzące w skład jednego pliku są zapisane na różnych dyskach macierzy, podczas pracy mogą mieć miejsce niezależne odczyty danych z poszczególnych dysków. Jednak w danej chwili może być wykonywana wyłącznie jedna operacja zapisu, gdyż wszystkie one wykorzystują jeden dysk zawierający dane kontrolne. Ogranicza to istotnie szybkość operacji zapisu danych do macierzy dyskowej RAID-4.



Rys. 6.31. Organizacja macierzy RAID-3

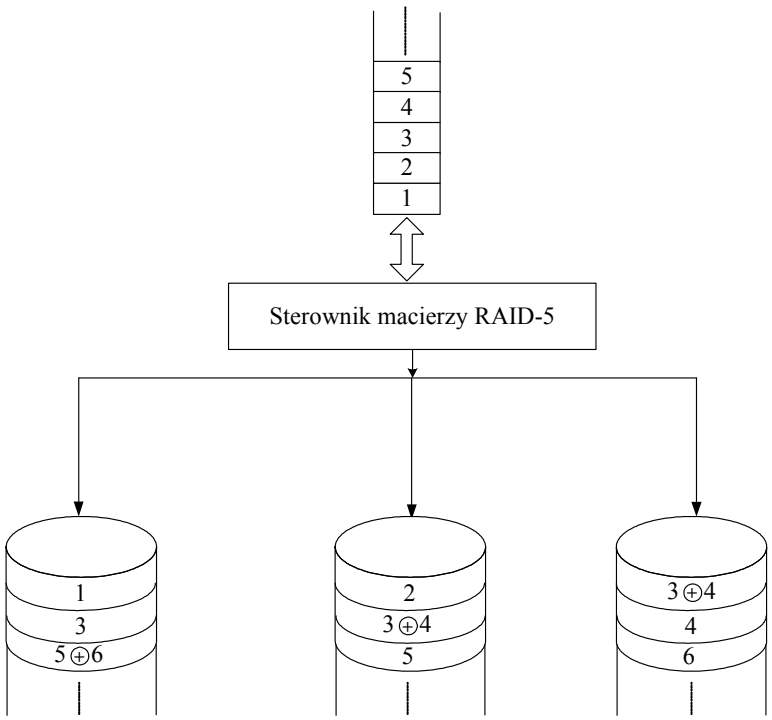
W macierzy dyskowej RAID-5 wykorzystuje się metodę analogiczną do RAID-4, jednak dane kontrolne nie są umieszczane na jednym dysku (jak to miało miejsce w RAID-4), lecz są równomiernie rozmieszczane na poszczególnych dyskach macierzy. Dzięki temu operacje zapisu danych kontrolnych dla poszczególnych bloków danych mogą odbywać się jednocześnie, co zwiększa szybkość zapisu do macierzy dyskowej. Organizację macierzy dyskowej RAID-5 przedstawiono na rys. 6.32.

Poza opisanymi rodzajami macierzy dyskowych stosowane są także pewne ich kombinacje. Przykładem może być macierz RAID-10, będąca kombinacją RAID-0 oraz RAID-1.

W tabeli 6.6 zebrano podstawowe charakterystyki omówionych macierzy dyskowych.

Tabela 6.6.

Konfiguracja RAID	**Redundancja (Nadmiarowość)**	**Odporność na uszkodzenia**	**Szybkość odczytu**	**Szybkość zapisu**
RAID-0	Nie	Nie	Podwyższona	Podwyższona
RAID-1	50%	Tak	Podwyższona	Obniżona
RAID-3, RAID-4, RAID-5	Maksymalnie 33%	Tak	Podwyższona	Obniżona (w różnym stopniu)
RAID-10	50%	Tak	Podwyższona	Podwyższona



Rys. 6.32. Organizacja macierzy RAID-5

Rozdział VII
KONCEPCJE PRZETWARZANIA ROZPROSZONEGO W SIECIOWYCH SYSTEMACH OPERACYJNYCH

Połączenie komputerów w sieć daje możliwość współpracy programów wykonywanych na różnych komputerach i w ten sposób umożliwia rozproszone przetwarzanie danych. Rozproszenie aplikacji w sieci daje wiele nowych możliwości w stosunku do aplikacji skupionych. Dzięki pracy równoległej osiągamy większą wydajność, wyższą odporność na awarie oraz skalowalność.

7.1. Modele usług sieciowych oraz aplikacji rozproszonych

Znaczna część programów – mimo że są sprowadzane z innego komputera w sieci – jest wykonywana na komputerze lokalnym, bez wykorzystywania zasobów innych komputerów przyłączonych do sieci. Takie aplikacje nie są aplikacjami sieciowymi. Jednak część aplikacji pracujących w sieci charakteryzuje się tym, że podczas realizacji wykorzystują zasoby innych komputerów w sieci. Takie aplikacje można określić mianem aplikacji sieciowych.

W sieci mogą być rozproszone nie tylko programy użytkowe, lecz także programy stanowiące część sieciowego systemu operacyjnego, wykonujące określone funkcje w systemie rozproszonym. Programy takie są określane mianem usług sieciowych.

Na organizację pracy aplikacji sieciowych mają wpływ następujące czynniki:

- sposób podziału aplikacji na części, wykonywane na różnych komputerach w sieci,
- wydzielenie specjalizowanych serwerów w sieci, wykonujących funkcje wykorzystywane przez wszystkie aplikacje,
- organizacja współpracy poszczególnych części aplikacji wykonywanych na różnych komputerach w sieci.

7.1.1. Sposób podziału aplikacji na części

W ogólnym przypadku aplikację można podzielić na sześć części funkcjonalnych:

- środki prezentacji danych (interfejs użytkownika),
- logika prezentacji danych, opisująca zasady oraz możliwe scenariusze współdziałania użytkownika z aplikacją (menu, listy elementów itp.),
- logika aplikacji – zbiór zasad przetwarzania danych,
- logika danych – organizacja danych w określonych bazach,

- wewnętrzne operacje bazy danych – operacje wykonywane przez serwer bazy danych,
- operacje plikowe – operacje standardowe w systemie plików.

Na podstawie tego modelu można zbudować cały szereg schematów rozłożenia części składowych aplikacji pomiędzy poszczególne komputery w sieci. Rozłożenie aplikacji między wieloma komputerami może znacznie podnieść jakość tej aplikacji (zwiększenie szybkości pracy, zwiększenie liczby równolegle obsługiwanych użytkowników itd.). Jednocześnie jednak czyni aplikację bardziej złożoną.

Do podstawowych schematów, powszechnie stosowanych w aplikacjach sieciowych, należą architektury dwu- oraz trójwarstwowe.

Architektura 2-warstwowa

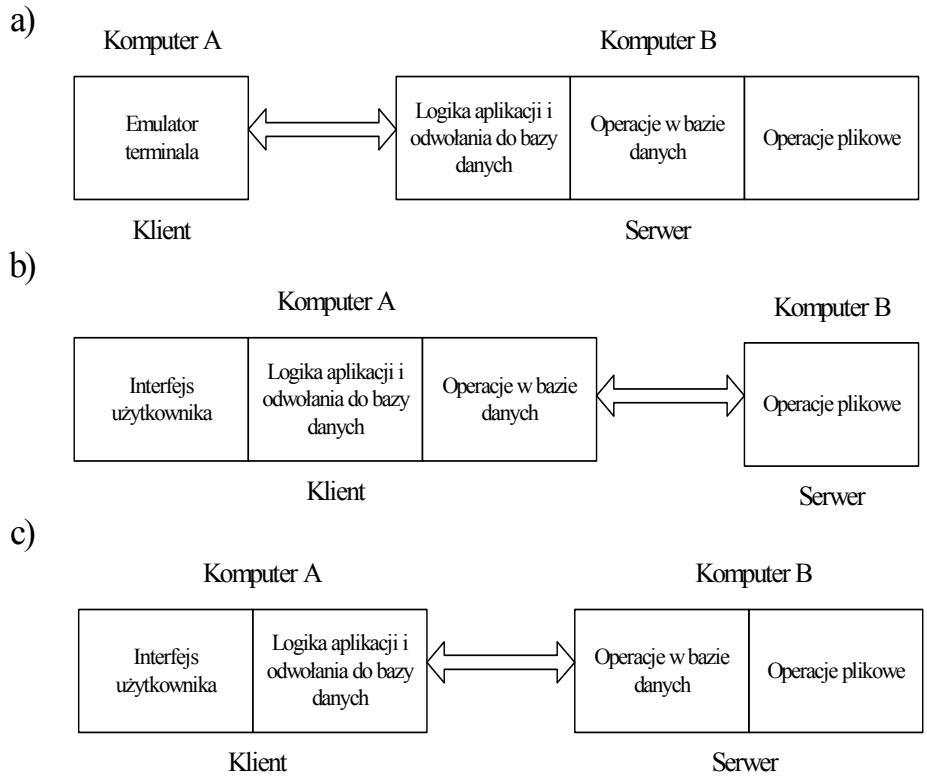
Architektura ta, określana mianem klient/serwer, charakteryzuje się rozłożeniem sześciu części funkcjonalnych aplikacji między dwa komputery, określane mianem serwera oraz klienta.

W architekturze z komputerem centralnym komputer użytkownika pracuje jako terminal, wykonujący wyłącznie funkcje prezentacji danych. Natomiast wszystkie pozostałe funkcje aplikacji są realizowane przez centralny komputer, pełniący rolę serwera (rys. 7.1a). W takim przypadku zasoby komputera klienckiego są wykorzystywane w niewielkim stopniu, gdyż wykonuje on wyłącznie następujące operacje: prezentacja danych (obsługa okien graficznych), obsługa sieci (przyjmowanie poleceń oraz danych od serwera, przesyłanie informacji o działaniach użytkownika – przyciśnięcie klawisza, współrzędne myszy itp.). Program wykonywany na komputerze klienckim określa się w takim przypadku mianem *emulatora terminala*. W istocie architektura ta naśladuje pracę komputera typu *mainfraim* z przyłączonymi terminalami z tą różnicą, że w miejscu terminali występują komputery podłączone nie poprzez lokalny interfejs, lecz poprzez sieć (lokalną lub globalną).

Główną wadą tej architektury jest niska skalowalność oraz wysoka wrażliwość na awarie. Podstawowym czynnikiem ograniczającym liczbę użytkowników (klientów) w systemie jest w tym przypadku moc obliczeniowa (wydajność) komputera centralnego, a awaria tego komputera uniemożliwia pracę wszystkim użytkownikom.

W architekturze z serwerem plików (rys. 7.1b) na komputerze klienckim są wykonywane wszystkie części funkcjonalne aplikacji z wyjątkiem operacji plikowych. Serwer w tym przypadku udostępnia wszystkim użytkownikom pliki przechowywane w systemie plików. Aplikacja rozproszona funkcjonująca w systemie z serwerem plików nieznacznie różni się od aplikacji działającej na lokalnym komputerze. Jedyną różnicą jest odwoływanie się aplikacji do plików przechowywanych na odległym komputerze. Aby w architekturze z serwerem plików było możliwe wykonywanie programów napisanych dla aplikacji lokalnych, do sieciowych systemów operacyjnych wprowadzono komponent

wykonujący operację przekierowania (ang. redirector), który przechwytuje odwołania do zdalnych plików (na podstawie specjalnej notacji dla nazw sieciowych, jak np. //server1/doc/plik1.txt) i kieruje te odwołania w sieć do odpowiedniego serwera plików.



Rys. 7.1.Systemy rozproszone w architekturze 2-warstwowej

Serwer plików realizuje jedną z najbardziej rozpowszechnionych usług sieciowych – sieciowy system plików. Pierwsze sieciowe systemy operacyjne (NetWare firmy Novell, IBM PC LAN Program, Microsoft MS-Net) wspierały dwie podstawowe usługi sieciowe: sieciowy system plików oraz usługę drukowania sieciowego, pozostawiając realizację pozostałych funkcji programistom aplikacji rozproszonych.

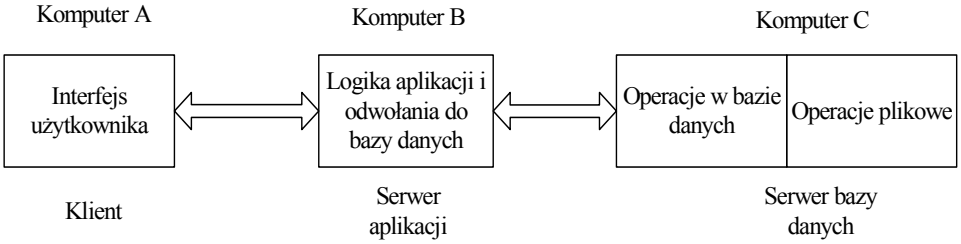
Architektura z serwerem plików charakteryzuje się wysoką skalowalnością, gdyż kolejni użytkownicy obciążają serwer w niewielkim stopniu. Jednak posiada także swoje wady:

- w wielu przypadkach jest generowany znaczny ruch w sieci (np. podczas operacji na danych może wystąpić konieczność przesłania całej bazy danych z serwera na komputer kliencki, na którym wykonywane są określone operacje, a następnie przesłanie bazy danych na serwer),
- komputer kliencki musi posiadać odpowiednią moc obliczeniową, gdyż wykonuje wszystkie operacje związane z przetwarzaniem danych.

Pozostałe warianty architektury dwuwarstwowej charakteryzują się bardziej równomiernym rozłożeniem funkcji między komputerem klienckim a serwerem. W najczęściej wykorzystywanym schemacie serwer wykonuje wewnętrzne operacje bazy danych oraz operacje plikowe (rys. 7.1c), podczas gdy na komputerze klienckim wykonywane są wszystkie pozostałe funkcje, związane z funkcjonowaniem konkretnej aplikacji. Dzięki temu, że serwer w takim przypadku wykonuje operacje niezależne od konkretnej aplikacji, funkcje realizowane przez niego mogą przyjmować postać usług sieciowych. Ze względu jednak na to, że funkcje związane z zarządzaniem bazami danych nie są potrzebne wielu aplikacjom sieciowym, więc usługi serwera bazy danych nie są najczęściej włączane do usług sieciowych, lecz udostępniane w postaci odrębnego oprogramowania spełniającego funkcje zarządzania bazami danych (serwery baz danych).

Architektura trójwarstwowa

Architektura ta pozwala na jeszcze korzystniejsze rozłożenie obciążenia pomiędzy poszczególnymi komputerami w sieci, a także umożliwia dalszą specjalizację serwerów. Za przykład architektury trójwarstwowej może służyć taka organizacja aplikacji, w której na komputerze klienckim znajduje się interfejs użytkownika oraz wykonywane są programy realizujące logikę prezentacji danych, jak również programy pełniące rolę interfejsu z kolejną warstwą – serwerem aplikacji (rys. 7.2).



Rys. 7.2. System rozproszony w architekturze 3-warstwowej

Na serwerze aplikacji realizowane są funkcje związane z logiką aplikacji oraz logiką przetwarzania danych, stanowiące najważniejszą część aplikacji. Warstwa środkowa wywołuje z kolei wewnętrzne operacje bazy danych, które realizowane są w warstwie serwera. Serwer bazy danych realizuje wewnętrzne operacje na danych, jak również operacje plikowe. Przykładem takiej architektury może być system zawierający trzy komputery połączone siecią:

- komputer kliencki, wykorzystujący przeglądarkę internetową (interfejs użytkownika),
- serwer aplikacji OAS (Oracle Application Server) firmy Oracle, realizujący funkcje przetwarzania danych z wykorzystaniem różnych środowisk programistycznych (Java, JSP, PHP, Servlety),
- serwer bazy danych Oracle 10g realizujący operacje związane z przechowywaniem, udostępnianiem i przetwarzaniem danych.

Moduły programowe realizujące funkcje warstwy pośredniej w architekturze trójwarstwowej określane są mianem *middleware* i mają także inne zastosowania niż bazy danych, np.:

- asynchroniczne przetwarzanie komunikatów (ang. Message Oriented Middleware – MOM),
- zdalne wywołanie procedury (ang. Remote Procedure Call – RPC),
- broker obiektów (ang. Object Request Broker – OBR).

Środki te poprawiają współdziałanie klientów z serwerami dzięki uporządkowaniu wywołań wielu klientów do szeregu serwerów, a także odgrywają rolę regulatorów rozkładających obciążenie między wiele serwerów. Serwer aplikacji powinien dysponować odpowiednią mocą obliczeniową, w przeciwnym wypadku będzie wąskim gardłem (ang. bottle neck) w systemie. Osiąga się to dzięki zastosowaniu komputerów wieloprocesorowych, a przede wszystkim komputerów klastrowych lub gridowych. Architektury takie charakteryzują się wysoką skalowalnością, gdyż istnieje możliwość znacznej rozbudowy systemu przy rosnących wymaganiach.

7.2. Mechanizm przekazywania komunikatów w systemach rozproszonych

Szczególnie ważnym zagadnieniem w systemach rozproszonych jest problem współdziałania procesów umieszczonych na różnych komputerach. Z problemem tym są związane dwa zagadnienia:

- stworzenie a następnie utrzymywanie kanału komunikacyjnego między procesami na różnych komputerach w sieci, zapewniającego dwukierunkowy transport danych,
- zapewnienie synchronizacji procesów realizowanych na różnych komputerach w sieci (często oddalonych od siebie o tysiące kilometrów).

Zasadniczym mechanizmem komunikacji rozproszonej jest przekazywanie komunikatów. Komunikat przekazywany od klienta do serwera zawiera najczęściej żądanie wykonania określonej usługi świadczonej przez serwer. Natomiast komunikat przekazywany przez serwer do klienta zawiera rezultat wykonania tej usługi. Np. klient na stacji roboczej żąda od serwera przekazania zawartości określonego katalogu w systemie plików. W odpowiedzi serwer przesyła klientowi komunikat zawierający listę plików i podkatalogów znajdujących się w danym katalogu.

Pod pojęciem komunikatu rozumiemy blok danych o określonym formacie, zrozumiałym tak dla klienta jak i serwera. Komunikat zawiera nagłówek o podanej długości oraz zbiór danych określonego typu o zmiennej długości. W nagłówku komunikatu umieszczone są następujące elementy:

- pole adresów, stanowiące zbiór symboli, jednoznacznie określających proces wysyłający oraz proces odbierający komunikat,
- numer komunikatu, niezbędny do jego identyfikacji w przypadku zgubienia oraz wystawiania duplikatów,
- struktura zawierająca pola o ściśle określonym znaczeniu, a mianowicie: pola typu danych, pola długości danych oraz pola samych danych. Pole typu określa typ przekazywanych danych (numeryczne całkowite, ciąg symboli itp.). Komunikat może zawierać szereg elementów zawierających opisane trzy pola.

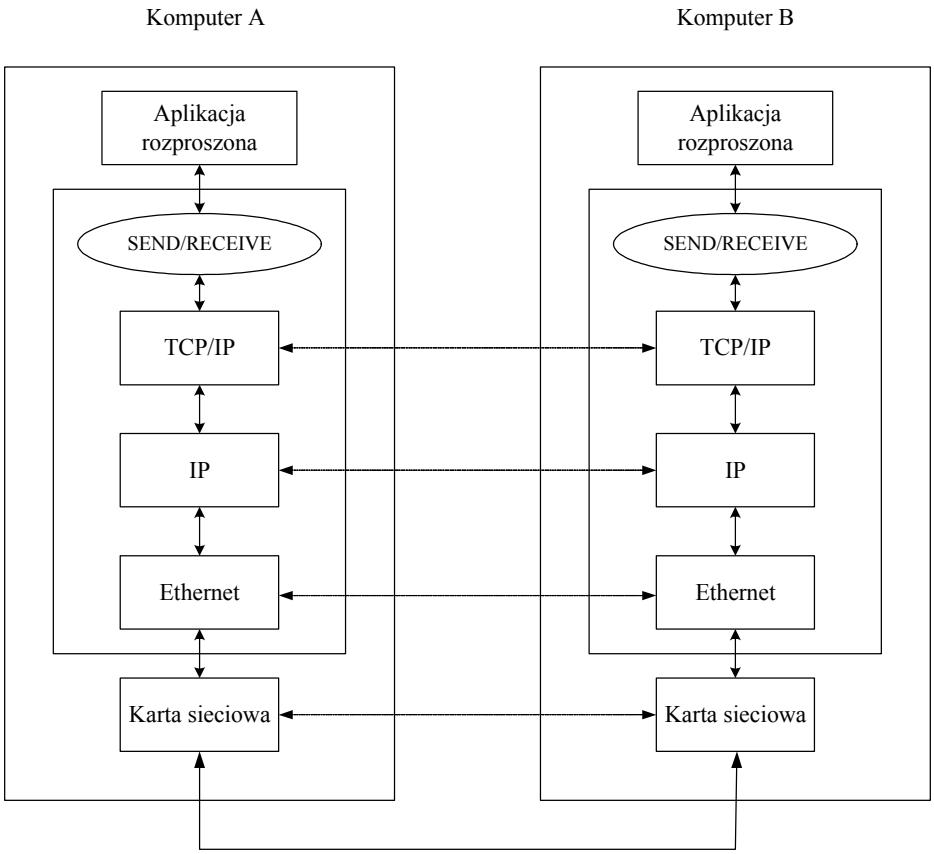
W każdym sieciowym systemie operacyjnym znajduje się podsystem przekazywania komunikatów, określany także mianem podsystemu transportowego, zawierający zbiór środków służących do organizacji współdziałania procesów w sieci. Celem tego podsystemu jest ukrycie szczegółów złożonych protokołów sieciowych przed programistą. Podsystem pozwala na współdziałanie procesów w sieci za pośrednictwem prostych prymitywów (pod pojęciem prymitywu rozumiemy elementarną funkcję realizowaną przez system operacyjny). Systemowe środki komunikacji międzyprocesowej w sieci mogą być sprowadzone do dwóch podstawowych prymitywów komunikacyjnych: *send* – do wysłania komunikatu oraz *receive* – do jego odbioru. Na bazie tych dwóch prymitywów mogą zostać stworzone bardziej złożone funkcje służące do komunikacji sieciowej, takie jak rozproszony system plików czy usługa wywołania zdalnej procedury.

Podsystem transportowy sieciowego systemu operacyjnego posiada zwykle bardzo złożoną strukturę, wynikającą z siedmiowarstwowego modelu OSI (ang. Open System Interconnection). W ten sposób podczas wykonania prymitywów *send* oraz *receive* mają miejsce wywołania wszystkich protokołów komunikacyjnych należących do warstw niższych. Na rys. 7.3 przedstawiono zasady przekazywania komunikatów na podstawie 4-warstwowego modelu sieci Internet.

Podczas realizacji prymitywów przekazywania komunikatów w systemie operacyjnym konieczna jest także odpowiedź na szereg pytań, jak np.:

- czy w sieci może być wielu odbiorców komunikatu, czy tylko jeden?
- czy jest konieczność zagwarantowania dostarczenia komunikatu?
- czy nadawca powinien oczekiwać na odpowiedź od odbiorcy nim rozpocznie wysyłanie innego komunikatu?
- jak nadawca i odbiorca oraz podsystem przekazywania komunikatów mają reagować na awarie kanału transmisyjnego?
- jak postępować gdy odbiorca nie jest gotowy do przyjęcia komunikatu?

Odpowiedzi na te pytania określają semantykę konkretnego protokołu przekazywania komunikatów.



Rys. 7.3. Przekazywanie komunikatów w 4-warstwowym modelu sieci Internet

7.2.1. Synchronizacja

Współdziałanie procesów w sieci i wymiana komunikatów wymagają synchronizacji. Z tego punktu widzenia prymitywy komunikacyjne można podzielić na *blokujące* (synchroniczne) oraz *nieblokujące* (asynchroniczne).

W przypadku blokującego prymitywu *send,* proces wysyłający komunikat zostaje zatrzymany do momentu otrzymania od odbiorcy potwierdzenia otrzymania komunikatu. Natomiast wywołanie blokującego prymitywu *receive* powoduje zatrzymanie procesu do momentu otrzymania komunikatu.

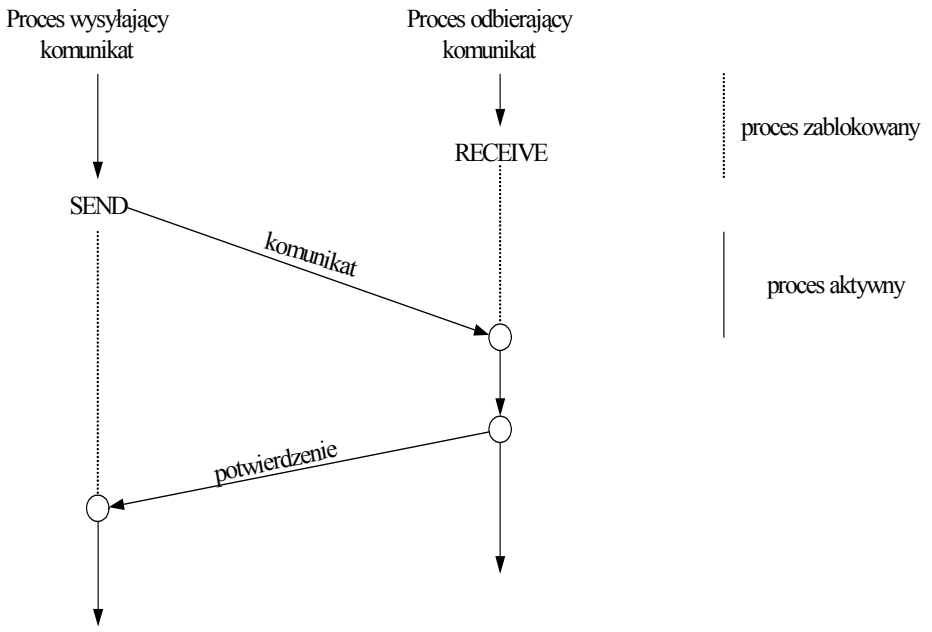
W przypadku nieblokujących prymitywów *send* oraz *receive* proces wywołujący kontynuuje swoją realizację natychmiast po uzyskaniu od jądra systemu operacyjnego informacji o adresie bufora, poprzez który będzie miało miejsce przekazywanie komunikatu.

Ważnym zagadnieniem przy wykorzystaniu nieblokującego prymitywu *receive* jest określenie sposobu poinformowania procesu-odbiorcy o nadejściu komunikatu. Stosowane są tu dwa rozwiązania:

- odpytywanie (ang. polling) polegające na cyklicznym wykonywaniu przez proces-odbiorcę prymitywu *test* sprawdzającego, czy w buforze znajduje się komunikat,
- przerwanie (ang. interrupt) informujące proces-odbiorcę o nadejściu komunikatu. Rozwiązanie to jest korzystniejsze, gdyż nie angażuje procesu-odbiorcy oraz systemu operacyjnego, jednak komplikuje program.

Podczas korzystania z blokującego prymitywu *send* może wystąpić sytuacja, gdy proces wysyłający komunikat zostanie zablokowany „na zawsze" na skutek nieprawidłowej pracy procesu-odbiorcy. Z tego powodu blokujący prymityw *send* musi mieć określony interwał czasu, po którym zakończy swoje działanie ze statusem „błąd" (ang. time-out). Analogiczna sytuacja ma miejsce w przypadku prymitywu *receive*.

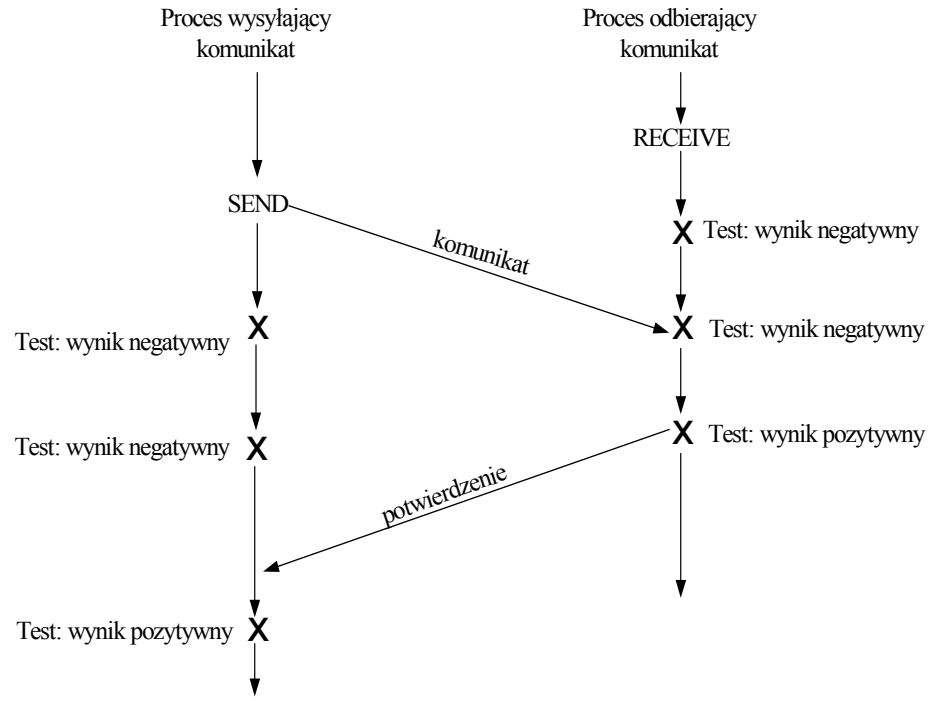
W przypadku gdy w obydwu procesach (nadawcy i odbiorcy) są wykorzystywane blokujące prymitywy *send* oraz *receive,* mówimy o synchronicznej współpracy procesów (rys. 7.4).



Rys. 7.4. Synchroniczna współpraca procesów

W przeciwnym wypadku współpraca odbywa się asynchronicznie (rys. 7.5). Z punktu widzenia programów użytkowych realizacja współpracy synchronicznej jest znacznie prostsza oraz bardziej niezawodna. Jednak charakteryzuje się pewnymi wadami, do których można zaliczyć ograniczoną równoległość realizacji aplikacji oraz możliwość wystąpienia blokad. Z zasady

systemy operacyjne udostępniają tak blokujące, jak i nieblokujące prymitywy *send* oraz *receive*.



Rys. 7.5. Asynchroniczna współpraca procesów

7.2.2. Sposoby adresacji

Mechanizm przekazywania komunikatów w sieci wymaga określenia adresów nadawcy i odbiorcy. Jedną z możliwości adresacji jest wykorzystanie numerów kart sieciowych zainstalowanych w poszczególnych komputerach. Taki sposób adresowania posiada jednak istotne ograniczenia. A mianowicie może być stosowany wyłącznie w sytuacji, gdy na danym węźle sieci jest wykonywany jeden proces biorący udział w przesyłaniu komunikatów. Ponadto w oparciu o numer karty sieciowej można przekazywać komunikaty wyłącznie w jednej sieci lokalnej. Do przekazywania komunikatów w rozbudowanych sieciach zawierających szereg podsieci, a w tym w sieci globalnej, jest niezbędny sposób adresowania określający numer węzła w sieci oraz numer podsieci.

Najszerzej współcześnie stosowanym rodzajem adresu jest adres w protokole IP (ang. Internet Protocol). Ten 32-bitowy adres jest podawany w postaci czterech liczb dziesiętnych z przedziału od 0 do 255, oddzielonych kropkami, np. 168.25.135.24. Stosowane są także inne sposoby adresowania węzła w sieci, jak np. adresy w protokołach IPX czy ATM.

Poza określeniem adresu węzła w sieci niezbędne jest także określenie procesu realizowanego na węźle, dla którego jest przeznaczony dany komunikat. Może to być identyfikator procesu określony przez system operacyjny podczas tworzenia tego procesu (takie rozwiązanie jest stosowane np. w systemie QNX). Znacznie bardziej uniwersalnym rozwiązaniem jest jednak określenie numeru usługi realizowanej na danym węźle sieci, a nie identyfikatora konkretnego procesu. W takim przypadku każdej usłudze przypisany zostaje jednoznaczny identyfikator. Rozwiązanie to ma jednak także istotne ograniczenie polegające na tym, że na danym węźle sieci może być realizowana wyłącznie jedna usługa o danym numerze. Przykładem takiego rozwiązania może być określenie numeru *portu* stosowane w protokołach TCP oraz UDP (np. usługa HTTP port nr 80, usługa FTP port nr 21 itd.). Dzięki takiemu rozwiązaniu proces wysyłający komunikat do serwera HTTP pracującego na odległym węźle sieci nie musi znać identyfikatora procesu serwera HTTP na tym komputerze. Wystarczy, że posłuży się numerem portu 80.

Numery portów stosowane w protokołach TCP oraz UDP są współcześnie najszerzej stosowanym sposobem określania usług podczas wymiany komunikatów w sieciowych systemach operacyjnych. Dla każdego portu system operacyjny tworzy bufor w pamięci operacyjnej, w którym są umieszczane komunikaty przeznaczone dla danego portu. W protokołach TCP i UDP numer portu jest liczbą dwubajtową bez znaku, co oznacza, że numery portów mogą być liczbami z przedziału od 0 do 65535, z czego numery z przedziału od 0 do 1023 zostały zarezerwowane dla usług standardowych.

Opisany schemat adresowania typu „komputer-proces" lub „komputer-usługa" jest współcześnie szeroko stosowany we wszystkich systemach operacyjnych. Jednak w sieci globalnej korzystanie z adresów IP jest mało elastyczne oraz niewygodne dla użytkownika. Łatwo sobie wyobrazić sytuację, w której np. firma Microsoft przenosi swój serwis WWW na komputer o innym adresie IP. W takim przypadku wszyscy użytkownicy tego serwisu są zmuszeni poznać i zapamiętać nowy adres. Ponadto w sieci może funkcjonować cały szereg serwerów WWW firmy Microsoft, a użytkownik jest kierowany do jednego z nich.

Podstawowym sposobem rozwiązania tych problemów i zwiększenia elastyczności oraz „przeźroczystości" adresowania jest zastosowanie nazw symbolicznych określających komputery w sieci oraz udostępniane usługi. Przykładem takiego podejścia jest charakterystyczne dla współczesnego Internetu pojęcie URL (ang. Universal Resource Locator), w którym adres składa się z symbolicznej nazwy węzła oraz symbolicznej nazwy usługi. Np. adres http://www.microsoft.com oznacza serwis WWW udostępniany na komputerze www.microsoft.com.

Wykorzystywanie adresów symbolicznych wymaga stworzenia w sieci usługi tłumaczenia nazw symbolicznych na adresy numeryczne. W tym celu mogą być stosowane dwa podejścia: rozgłoszeniowe oraz centralna usługa nazw. Metoda rozgłoszeniowa jest wygodna do zastosowania w sieciach lokalnych, w których wszystkie sieciowe technologie niższego poziomu, takie

jak Ethernet, Token Ring, FDDI itp. stosują adresy rozgłoszeniowe, a przepustowość sieci jest wystarczająca dla przesyłania komunikatów rozgłoszeniowych. Na metodzie rozgłoszeniowej były oparte wszystkie usługi systemu operacyjnego NetWare do wersji 4, który w swoim czasie był wzorcem przeźroczystości dla użytkowników. W tym przypadku serwer periodycznie wysyła rozgłoszeniowo komunikaty o numerycznych adresach odpowiadających nazwie serwera oraz nazwom udostępnianych usług. Rozgłoszeniowy komunikat może także wysłać klient w celu uzyskania informacji o adresie serwera obsługującego określoną usługę. Jeżeli serwer taki znajduje się w sieci, to odpowiada na zapytanie klienta swoim adresem numerycznym.

Rozgłoszeniowy mechanizm wyznaczania adresów nie nadaje się do zastosowania w sieci globalnej ze względu na dużą liczbę klientów i serwerów oraz mniejszą przepustowość sieci w stosunku do sieci lokalnych. W takim przypadku stosowane jest inne podejście, oparte na specjalizowanych serwerach zawierających bazy danych nazw symbolicznych i odpowiadających im adresów numerycznych. Serwery te tworzą rozproszoną usługę nazw obsługującą klientów. Powszechnie znanym przykładem takiej usługi jest usługa DNS (ang. Domain Name Service). Usługa ta pozwala na obsługę w czasie rzeczywistym wielu klientów sieci Internet, którzy odwołują się do zasobów serwerów z wykorzystaniem nazw symbolicznych.

7.2.3. Systemowe środki komunikacji międzyprocesowej

W dotychczasowych analizach zakładano, że wysłany komunikat z pewnością dociera do odbiorcy. W rzeczywistości jednak nie ma takich gwarancji. Problem zapewnienia gwarancji prawidłowego przekazania komunikatu może być rozwiązany na trzy sposoby.

Pierwsze z rozwiązań, określane mianem datagramowego, polega na tym, że system operacyjny nie bierze na siebie odpowiedzialności za dostarczenie komunikatu do odbiorcy. W takim przypadku problem niezawodności dostarczenia komunikatu musi być rozwiązany przez programistę aplikacji.

Drugi sposób polega na tym, że jądro systemu operacyjnego komputera odbierającego komunikat przesyła do jądra systemu nadawcy komunikat będący potwierdzeniem odbioru. Do momentu odbioru potwierdzenia proces wysyłający komunikat zostaje zablokowany. W tym przypadku problem synchronizacji procesów nadawcy i odbiorcy jest rozwiązany przez systemy operacyjne funkcjonujące na obydwu komputerach, podczas gdy procesy nadawcy i odbiorcy „nie widzą" działań związanych z synchronizacją i potwierdzaniem odbioru komunikatów.

Trzeci sposób jest typowy dla architektury klient-serwer i polega na wykorzystaniu odpowiedzi jako potwierdzenia otrzymania komunikatu.

Proces wysyłający komunikat zostaje zablokowany do momentu otrzymania odpowiedzi, a jeżeli odpowiedź nie nadchodzi w określonym czasie to system operacyjny ponownie wysyła komunikat.

Niezawodne przekazywanie komunikatów można rozumieć nie tylko jako gwarancję dostarczenia poszczególnych komunikatów, lecz także zapewnienie, że dane te zostaną odebrane przez proces-odbiorcę w takiej kolejności, w jakiej zostały wysłane (protokoły sieciowe często nie gwarantują, że komunikaty zostaną tak dostarczone do odbiorcy).

Podsystemy wymiany komunikatów systemów operacyjnych dostarczają prymitywów, tak niezawodnych jak i zawodnych. Pozwala to programiście aplikacji na korzystanie z takiego prymitywu, jaki jest w danym przypadku najbardziej odpowiedni. Podczas przekazywania danych o znacznym rozmiarze, transportowanych przez sieć w wielu komunikatach (w protokołach sieciowych z reguły funkcjonują ograniczenia co do rozmiaru pola danych w komunikacie), znacznie korzystniejsze jest zastosowanie niezawodnej metody przekazywania danych, zapewniającej ich uporządkowanie. Natomiast podczas przekazywania krótkich komunikatów w sieci lokalnej, charakteryzującej się wysoką niezawodnością, korzystne będzie zastosowanie prymitywów zawodnych, niekontrolujących poprawności procesu przesyłania danych.

W przypadku gdy podsystem wymiany komunikatów systemu operacyjnego wykorzystuje protokół IP, to w celu niezawodnego przekazywania komunikatów stosowany jest protokół TCP (ang. Transfer Control Protocol), określany mianem protokołu połączeniowego. Protokół ten daje gwarancję dostarczenia komunikatu oraz zapewnia, że dane zostaną odebrane w takiej kolejności, w jakiej zostały wysłane. Natomiast zawodne przekazywanie komunikatów jest realizowane przy pomocy protokołu bezpołączeniowego UDP (ang. Universal Datagram Protocol), zapewniającego szybkie przekazywanie komunikatów bez zapewnienia gwarancji ich dostarczenia. Analogicznie, w sieciach zarządzanych przez system NetWare, niezawodne przekazywanie komunikatów odbywa się z zastosowaniem protokołu SPX, natomiast zawodne – IPX.

7.3. Mechanizm gniazd (ang. sockets) w systemie UNIX

Gniazda po raz pierwszy pojawiły się w wersji 4.3 systemu BSD UNIX (Berkeley Software Distribution UNIX), a następnie rozpowszechniły się we wszystkich wersjach systemu UNIX, a także Windows (Windows Sockets – WinSock). Mechanizm gniazd zapewnia wygodny oraz uniwersalny interfejs wymiany komunikatów, przeznaczony dla aplikacji rozproszonych. Uniwersalność tego mechanizmu zapewniają następujące koncepcje:

- niezależność od protokołów i technologii stosowanych w niższych warstwach. W tym celu zostało wprowadzone pojęcie domeny komunikacyjnej, posiadającej zbiór właściwości komunikacyjnych określającej sposób tworzenia nazw węzłów sieci oraz zasobów, charakterystyki połączeń sieciowych, sposoby synchronizacji procesów itp. Podstawową domeną jest domena internetowa z protokołami TCP/IP,
- wprowadzenie abstrakcyjnego punktu połączenia nazwanego gniazdem. Gniazdo jest punktem, poprzez który przekazywane są komunikaty między danym węzłem a siecią. Każdy proces posługuje się własnym gniazdem. Połączenie między dwoma procesami jest realizowane poprzez parę gniazd,
- gniazdo posiada przydzielony adres, który w domenie internetowej jest parą: adres IP, port,
- dla każdej domeny komunikacyjnej mogą istnieć różne typy gniazd (gniazda datagramowe, gniazda strumieniowe).

Mechanizm gniazd wykorzystuje następujące prymitywy, zrealizowane w postaci wywołań systemowych:

Stworzenie gniazda:

```
deskr = secket(domena, typ, protokół)
```

Gniazdo należy stworzyć w procesie nim zaczniemy z niego korzystać. Parametr `domena` określa jedną z domen, z których możemy korzystać. Standardowo w SO UNIX występują dwie domeny: internetowa – określona stałą AF_INET oraz uniksowa – stałą AF_UX. Parametr `typ` określa typ gniazda (stała SOCK_DGRAM – gniazdo datagramowe, SOCK_STREAM – strumieniowe). Parametr `protokół` może przyjąć wartość 0 (w takim przypadku system sam przyjmie odpowiedni protokół). Przedstawione stałe są zdefiniowane w pliku nagłówkowym `socket.h`. W procesie gniazdo jest reprezentowane przez zmienną całkowitoliczbową `deskr`, określającą numer pozycji w tablicy deskryptorów otwartych plików procesu.

Związanie gniazda z adresem

```
ok = bind(deskr, adres, dl_adr)
```

Przypisywanie adresu danemu gniazdu jest niezbędne jedynie w przypadku, gdy będzie ono przyjmować komunikaty. Dla domeny internetowej parametr *adres* zawiera adres IP oraz port. Parametr `dl_adr` określa długość adresu. Funkcja `bind` zwraca wartość 0 (operacja poprawna) lub –1 (operacja zakończona niepomyślnie).

Nasłuchiwanie (oczekiwanie na utworzenie połączenia)

```
listen(deskr, dl_kol)
```

Funkcja realizuje oczekiwanie serwera na nawiązanie połączenia przez klienta. Parametrami funkcji są: deskryptor gniazda oraz maksymalna długość kolejki

procesów pełniących rolę klientów, oczekujących na nawiązanie połączenia. Jeśli kolejka procesów oczekujących na połączenie jest pusta, to proces zostaje zablokowany do momentu pojawienia się w kolejce procesu klienta.

Inicjacja połączenia

```
ok = connect (deskr, adres_serw, dl_adr)
```

Funkcja `connect` jest wykorzystywana wyłącznie dla protokołu połączeniowego (gniazdo strumieniowe), który wymaga ustanowienia połączenia. Funkcję tę wykonuje proces klienta, który inicjuje nawiązanie połączenia. Parametrami funkcji są: deskryptor gniazda, adres procesu-serwera oraz długość adresu. Po ustanowieniu połączenia funkcja zwraca wartość 0, natomiast w przypadku wystąpienia błędu - wartość –1. Przesłanie komunikatu z zastosowaniem zainicjowanego połączenia odbywa się za pomocą funkcji `write`, analogicznie jak zapis danych do pliku.

Zaakceptowanie połączenia

```
deskr1 = accept(deskr, adres_kl, dl_adr)
```

Funkcję `accept` wykonuje proces serwera po przyjęciu zgłoszenia od klienta (odpowiedź na inicjację połączenia). Funkcja tworzy nowe gniazdo identyfikowane deskryptorem `deskr1`, które służy do przekazywania komunikatów między klientem i serwerem.

Wysłanie komunikatu przez ustanowione połączenie

```
nbytes = write(deskr1, bufor, dl_kom)
```

Funkcja `write` przekazuje komunikat zapisany w buforze `bufor` do odbiorcy przez ustanowione wcześniej połączenie.

Odbiór komunikatu przez ustanowione połączenie

```
nbytes = read(deskr1, bufor, dl_kom)
```

Funkcja `read` odczytuje komunikat przekazany przez ustanowione wcześniej połączenie do bufora `bufor`. W przypadku gdy bufor jest pusty, proces zostaje zablokowany aż do momentu nadejścia komunikatu.

Wysłanie komunikatu bez ustanowienia połączenia

```
sendto(deskr, kom, adres_odb)
```

Funkcję `sendto` wykorzystuje się do przesłania komunikatu `kom` do odbiorcy `adres_odb` z wykorzystaniem gniazda datagramowego (SOCK_DGRAM).

Odbiór komunikatu bez ustanowienia połączenia

```
nbytes = recvfrom(deskr, kom, adres_nad)
```

Funkcję `recvfrom` wykorzystuje się do odbioru komunikatu od procesu `adres_nad`. Komunikat zostaje zapisany w buforze `kom`. W przypadku gdy komunikat nie został wysłany, proces jest blokowany i oczekuje na nadejście komunikatu.

Dla komunikacji z zastosowaniem gniazd bezpołączeniowych (SOCK_DGRAM) fragment programu wysyłającego komunikat może być następujący:

```
deskr = socket(AF_INET, SOCK_DGRAM, 0);
...
bind(deskr, adr_nad, dl_adr);
sendto(deskr, kom, adr_odb);
...
close(deskr);
```

natomiast programu odbierającego komunikat:

```
deskr = socket(AF_INET, SOCK_DGRAM, 0);
...
bind(deskr, adr_odb, dl_adr);
...
nbytes = recvfrom(deskr, kom, dl_adr);
...
close(deskr);
```

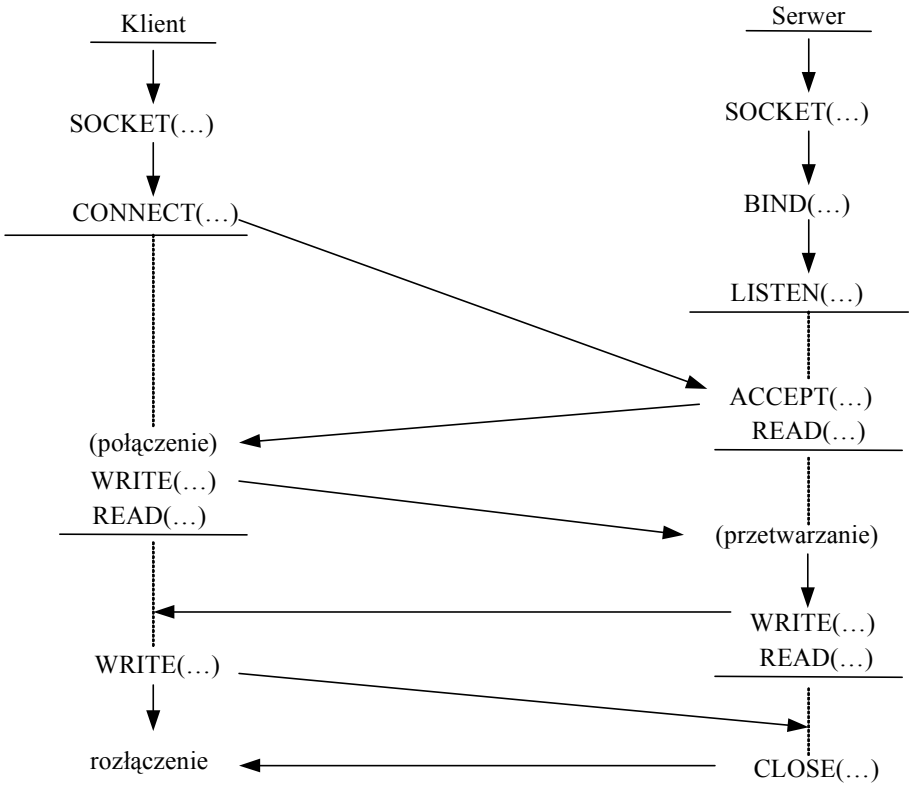
Dla komunikacji z zastosowaniem gniazd połączeniowych (SOCK_STREAM) fragment programu serwera może być następujący:

```
deskr = socket(AF_INET, SOCK_STREAM, 0);
...
bind(deskr, adr_serw, dl_adr);
listen(deskr, dl_kol);
...
deskr_1 = accept(deskr, adr_klient, dl_adr);
...
nbytes = read(deskr_1, bufor, dl_kom);
...
nbytes = read(deskr_1, bufor, dl_kom);
...
close(deskr);
```

natomiast programu klienta:

```
deskr = socket(AF_INET, SOCK_STREAM, 0);
...
connect(deskr, adr_serw, dl_adr);
...
write(deskr, kom, dl_kom);
...
write(deskr, kom, dl_kom);
...
close(deskr);
```

Na rys. 7.6 przedstawiono schemat komunikacji przy zastosowaniu gniazd połączeniowych (podkreślono funkcje blokujące, a linią kropkową zaznaczono procesy zablokowane).



Rys. 7.6. Komunikacja międzyprocesowa z zastosowaniem gniazd połączeniowych

7.4. Wywoływanie zdalnych procedur

Mechanizm RPC (ang. Remote Procedure Call) stanowi wyższą warstwę programową, nadbudowaną nad podsystemem wymiany komunikatów. W szeregu przypadków mechanizm ten pozwala na znacznie wygodniejszą i przeźroczystą organizację współdziałania programów w sieci. Istotną wadą tego rozwiązania jest jednak fakt, że nie jest to mechanizm uniwersalny.

7.4.1. Koncepcja zdalnego wywołania procedury

Koncepcja RPC jest rozszerzeniem koncepcji przekazania sterowania oraz danych wewnątrz programu wykonywanego na jednym komputerze na koncepcję przekazania sterowania i danych na inny komputer za pośrednictwem sieci. Mechanizm RPC w sposób istotny zwiększa możliwości rozproszonego

przetwarzania danych i dobrze odpowiada dewizie „Sieć to komputer”. Największą efektywność RPC uzyskuje się w tych aplikacjach, w których istnieją interaktywne połączenia między odległymi komponentami systemu, przy krótkim czasie odpowiedzi poszczególnych komputerów oraz niewielkich zbiorach przekazywanych danych. Aplikacje takie określa się mianem zorientowanych na RPC.

Do charakterystycznych cech procesu lokalnego wywołania procedury należy zaliczyć:

- asymetrię – jeden ze współpracujących programów jest stroną inicjującą współpracę,
- synchronizację – wykonywanie programu wywołującego procedurę zostaje zawieszone na czas wykonania procedury.

Realizacja wywołania procedury na odległym komputerze jest znacznie bardziej złożona od wywołania procedury lokalnej. Ponieważ procedura wywołująca oraz wywoływana są wykonywane na różnych komputerach, więc posiadają odrębne przestrzenie adresowe. Jest to źródłem wielu problemów podczas przekazywania parametrów oraz wyników między programami, szczególnie w przypadku, gdy odległe komputery są zarządzane różnymi systemami operacyjnymi. Ze względu na odseparowane przestrzenie adresowe obydwu programów przekazywane parametry nie mogą być wskaźnikami (adresami) danych, lecz wyłącznie skopiowanymi z jednego komputera na drugi wartościami.

Ponieważ w mechanizmie RPC biorą udział co najmniej dwa procesy wykonywane na różnych komputerach, więc należy określić zachowanie się systemu w różnych sytuacjach awaryjnych, a przede wszystkim:

- program, który wywołał zdalną procedurę, awaryjnie kończy działanie, podczas gdy wywołana procedura jest prawidłowo realizowana i chce przekazać wynik,
- zdalna procedura awaryjnie kończy działanie, podczas gdy program wywołujący oczekuje na wynik.

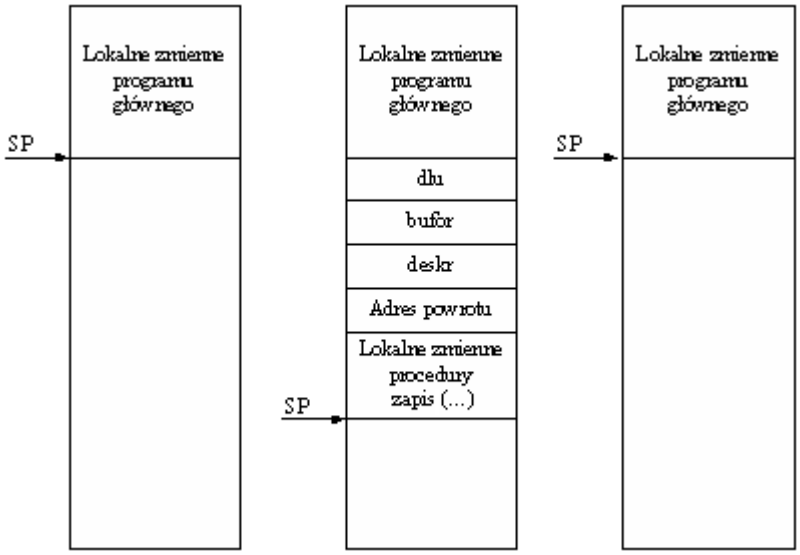
Podczas zdalnego wywołania podprogramu może wystąpić wiele problemów związanych z różnorodnością języków programowania oraz środowisk systemowych, a w szczególności z różnorodnością struktur danych oraz struktur wywołań podprogramów.

Nim szczegółowo omówimy proces wywołania zdalnej procedury, prześledźmy proces wywołania lokalnej procedury w języku C. Dla przykładu rozpatrzmy wywołanie funkcji:

```
l = zapis(deskr, bufor, dlu);
```

zapisującej dane znajdujące się w tablicy znaków `bufor` do pliku określonego przez deskryptor `deskr` (zmienna `dlu` określa liczbę zapisywanych bajtów). Przed wywołaniem funkcji `zapis` program główny odkłada na stos przekazywane parametry (w odwrotnej kolejności, poczynając od `dlu`), a następnie wykonuje skok do podprogramu umieszczonego w pamięci pod symbolicznym adresem `zapis`. W przypadku parametru `bufor` na stos jest

odkładany adres (wskaźnik) tablicy znaków `bufor`, a nie cała tablica. Podczas realizacji funkcji `zapis` zostaje wykonane wywołanie systemowe `write` (zapis danych do pliku). Analogicznie jak podczas wywołania podprogramu `zapis`, parametry dla tego wywołania zostają wcześniej odłożone na stos (ponieważ systemowa funkcja `write` jest wykonywana w trybie uprzywilejowanym, podczas wywołania systemowego następuje przepisanie danych ze stosu dla trybu użytkownika na stos dla trybu uprzywilejowanego). Po wykonaniu funkcji `zapis` wynik działania `l` zostaje zapisany do rejestru procesora, ze stosu pobierany jest adres powrotu do programu głównego i realizowany jest skok do programu głównego, który zdejmuje ze stosu odłożone wcześniej parametry. Na rys. 7.7 przedstawiono stan stosu dla trybu użytkownika przed wywołaniem funkcji `zapis`, podczas wykonywania funkcji `zapis` oraz po powrocie do programu głównego.



Rys. 7.7. Przekazywanie parametrów przez stos podczas wywołania procedury lokalnej w języku C

Mechanizm RPC został zaprojektowany tak, aby wywołanie zdalnej procedury odbywało się analogicznie do wywołania procedury lokalnej, co oznacza, że mechanizm ten jest przeźroczysty dla programisty (programista piszący program wywołujący procedurę zdalną postępuje analogicznie, jakby wywoływał procedurę lokalną). Mechanizm RPC jest realizowany następująco: gdy wywoływana procedura jest rzeczywiście odległa, do biblioteki procedur, w miejsce procedury lokalnej, wprowadza się specjalną wersję procedury określaną mianem stopki klienta (ang. client stub). Na komputerze odległym, pełniącym rolę serwera procedur, musi zostać umieszczony kod procedury, która będzie zdalnie wywoływana, oraz tzw. stopka serwera (server stub). Zadaniem stopek jest organizacja przekazywania parametrów do wywoływanej

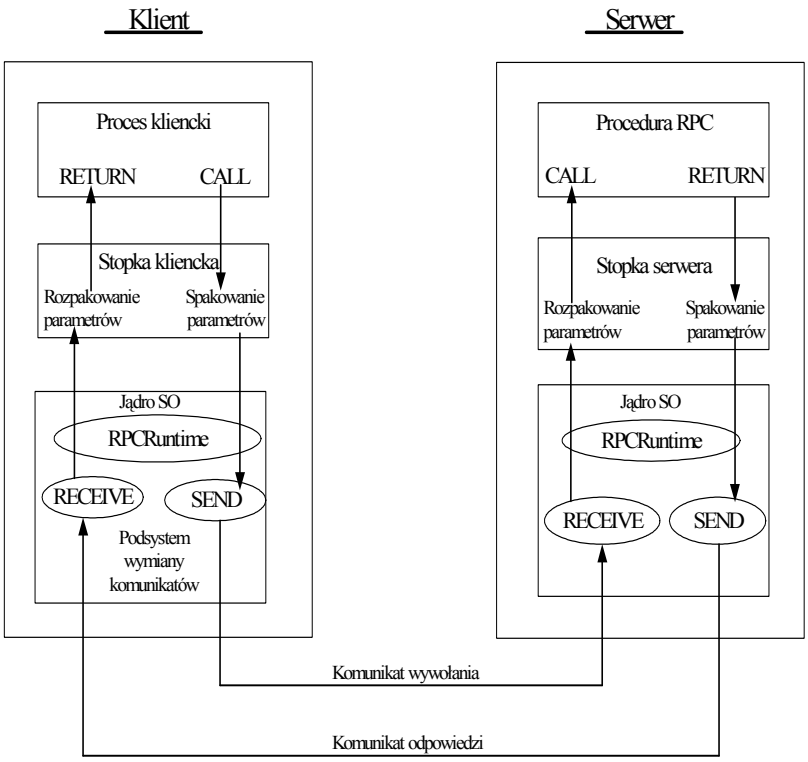
procedury oraz wyników do programu wywołującego. Do przekazywania danych między programem wywołującym a zdalną procedurą za pośrednictwem sieci jest wykorzystywany mechanizm wymiany komunikatów, z zastosowaniem prymitywów *send* oraz *receive*. W wielu przypadkach w jądrze systemu operacyjnego zostaje wydzielony moduł określany mianem RPCRuntime, służący do organizacji łączności stopek z prymitywami przekazywania komunikatów.

W programie wywołującym zdalną procedurę następuje wywołanie klienckiej stopki z przekazaniem parametrów przez stos, analogicznie jak podczas wywołania procedury lokalnej. Podczas wykonania stopki następuje utworzenie i przesłanie do serwera procedur komunikatu zawierającego nazwę wywoływanej procedury wraz z parametrami (rys. 7.8). Po odbiorze komunikatu przez serwer procedur jądro systemu operacyjnego wywołuje stopkę, która pobiera z komunikatu przekazane parametry i wywołuje właściwą procedurę z przekazaniem do niej parametrów poprzez stos. Po zakończeniu procedury stopka tworzy komunikat, umieszczając w nim wynik działania procedury, który następnie zostaje wysłany do komputera klienckiego za pomocą prymitywu *send*. Dzięki takiemu rozwiązaniu program wywołujący zdalną procedurę, jak również sama procedura, są napisane tak, jakby miały być wykonywane na jednym komputerze.

7.4.2. Generowanie stopek

Stopki mogą być generowane ręcznie przez programistę lub automatycznie. W przypadku generowania ręcznego, programista ma dużą swobodę w wyborze sposobu przekazywania parametrów wywołania oraz użycia prymitywów do przesyłania komunikatów. Automatyczny sposób generowania stopek jest oparty na zastosowaniu specjalnego języka definicji interfejsu (ang. Interface Definition Language – IDL), przy pomocy którego programista opisuje interfejs między klientem i serwerem RPC. Opis zawiera listę nazw procedur, które mogą być wywoływane przez klientów, oraz listę typów argumentów i wyników tych procedur. W oparciu o informacje zawarte w definicji interfejsu stopki kontrolują typy przekazywanych argumentów oraz generują wywołanie odległej procedury. Ponadto definicja interfejsu zawiera szereg dodatkowych informacji niezbędnych do optymalizacji współpracy stopek. I tak np. każdy parametr jest zaznaczony jako wejściowy, wyjściowy lub wejściowy i wyjściowy.

Opracowana przez programistę definicja interfejsu zostaje następnie poddana kompilacji przy pomocy kompilatora IDL. Kompilator IDL tworzy moduły wynikowe stopek klienckiej oraz serwerowej, a także generuje odpowiednie pliki nagłówkowe zawierające definicje typów procedur oraz ich argumentów (dla konkretnego języka programowania, a przede wszystkim języka C). Uzyskane moduły interfejsu mogą być następnie dołączone do programu użytkowego i wspólnie skompilowane w celu uzyskania programu wynikowego.



Rys. 7.8. Wykonanie zdalnego wywołania procedury

7.4.3. Format komunikatów RPC

Mechanizm RPC wykorzystuje dwa typy komunikatów: komunikaty-wywołania oraz komunikaty-odpowiedzi. Przy pomocy komunikatu-wywołania program kliencki wysyła do serwera RPC żądanie wykonania odległej procedury oraz przekazuje parametry, natomiast przy pomocy komunikatu-odpowiedzi serwer RPC przekazuje do klienta wynik działania odległej procedury. Przy pomocy tych komunikatów jest realizowany protokół RPC, określający sposób współdziałania klienta i serwera RPC. Protokół RPC jest niezależny od protokołów transportowych, przy pomocy których przekazywane są przez sieć komunikaty RPC. Typowy format komunikatu wywołania RPC posiada następujące pola:

- identyfikator komunikatu,
- typ komunikatu,
- identyfikator klienta,
- identyfikator procedury odległej:
  - numer programu,
  - numer wersji,
  - numer procedury,
  - argumenty.

Natomiast komunikat odpowiedzi RPC:

- identyfikator komunikatu,
- typ komunikatu,
- status odpowiedzi (błąd),
- wynik lub przyczyna błędu.

Pole określające typ komunikatu pozwala na rozróżnienie komunikatu-wywołania od komunikatu-odpowiedzi. Poszczególne procedury na serwerze RPC są identyfikowane numerem przydzielanym przez kompilator IDL (nie wykorzystuje się nazw procedur). Pole argumentów posiada zmienną długość, zależną od liczby oraz typów przekazywanych parametrów. W polu identyfikatora komunikatu umieszczona zostaje liczba kolejna identyfikująca dany komunikat. Pozwala ona klientowi na powiązanie uzyskanej odpowiedzi z wcześniejszym wywołaniem w przypadku, gdy odpowiedzi od serwera przychodzą w innej kolejności niż zostały wysłane. Identyfikator klienta jest niezbędny serwerowi RPC aby odpowiedź została wysłana do właściwego klienta. Pole statusu oraz pole wyniku w komunikatach-odpowiedziach pozwalają serwerowi powiadomić klienta o prawidłowym wykonaniu odległej procedury lub, w przypadku wystąpienia błędu podczas realizacji procedury, o kodzie błędu.

W celu zapewnienia stabilnej pracy serwera i klientów RPC jest konieczna obsługa sytuacji związanych z zagubieniem komunikatów, wynikającym z nieprawidłowego funkcjonowania sieci lub nieprawidłowego działania systemu operacyjnego. Protokół RPC wykorzystuje w takich sytuacjach mechanizm time-out z ponownym przekazaniem komunikatów.

7.4.4. Powiązanie klienta z serwerem

Istotnym zagadnieniem dla funkcjonowania mechanizmu RPC jest sposób, w jaki klient określa adres serwera, do którego będzie mógł przesłać komunikat-wywołanie. Procedura określająca powiązanie klienta z serwerem RPC określana jest mianem łączenia (ang. binding). Metody łączenia stosowane w różnych realizacjach RPC różnią się:

- sposobem określenia serwera, z którym chciałby być związany klient,
- sposobem określenia sieciowego adresu serwera RPC niezbędnego dla klienta,
- etapem, na którym zachodzi łączenie.

Metoda łączenia jest ściśle związana z przyjętą metodą określenia nazwy serwera. W najprostszym przypadku (tzw. metoda statyczna) nazwę lub adres serwera RPC podaje się w jawnej postaci, jako parametr stopki klienckiej lub serwera RPC. Przykładem takiego rozwiązania może być zastosowanie adresu IP komputera, na którym pracuje serwer RPC wraz z numerem portu, poprzez który przekazywane są zdalne wywołania procedur. Zasadniczą wadą takiego rozwiązania jest brak przeźroczystości dla aplikacji klienckiej oraz mała elastyczność systemu. Przy zmianie adresu serwera RPC należy dokonać

odpowiednich zmian w komputerach klienckich. Ponadto system taki nie może działać w sytuacji, gdy w sieci funkcjonuje szereg serwerów RPC, a wywołanie procedury jest kierowane do serwera najmniej obciążonego w danej chwili. Mimo przedstawionych wad rozwiązanie takie jest bardzo często stosowane ze względu na swoją prostotę.

Poza metodami statycznymi wiązania klienta z serwerem stosowane są także metody dynamiczne, niewymagające od klienta dokładnego określenia adresu serwera RPC, gdyż wyszukiwanie odpowiedniego serwera RPC jest wykonywane samoczynnie.

LITERATURA

A. Silberschatz, P. B. Galvin – Podstawy systemów operacyjnych. Warszawa, WNT. 2000.

M. J. Bach – Budowa systemu operacyjnego UNIX. Warszawa, WNT. 1995.

A. S. Tanenbaum – Rozproszone systemy operacyjne. Warszawa, PWN. 1997.

M. Beck, H. Bohme, M. Dziadzka, U. Knitz, R. Magnus, D. Verworner – LINUX kernel. Jądro systemu. Warszawa, Mikom. 1999.

A. M. Lister, R. D. Eager – Wprowadzenie do systemów operacyjnych. Warszawa, WNT. 1994.

D. A. Solomon – Inside Windows NT. Second Edition. Redmont, Washington, Microsoft Press. 1998.