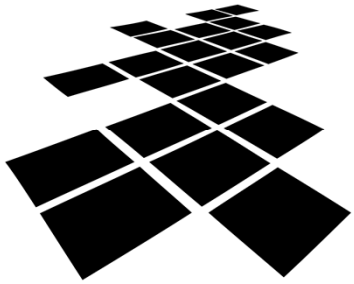
Systemy operacyjne

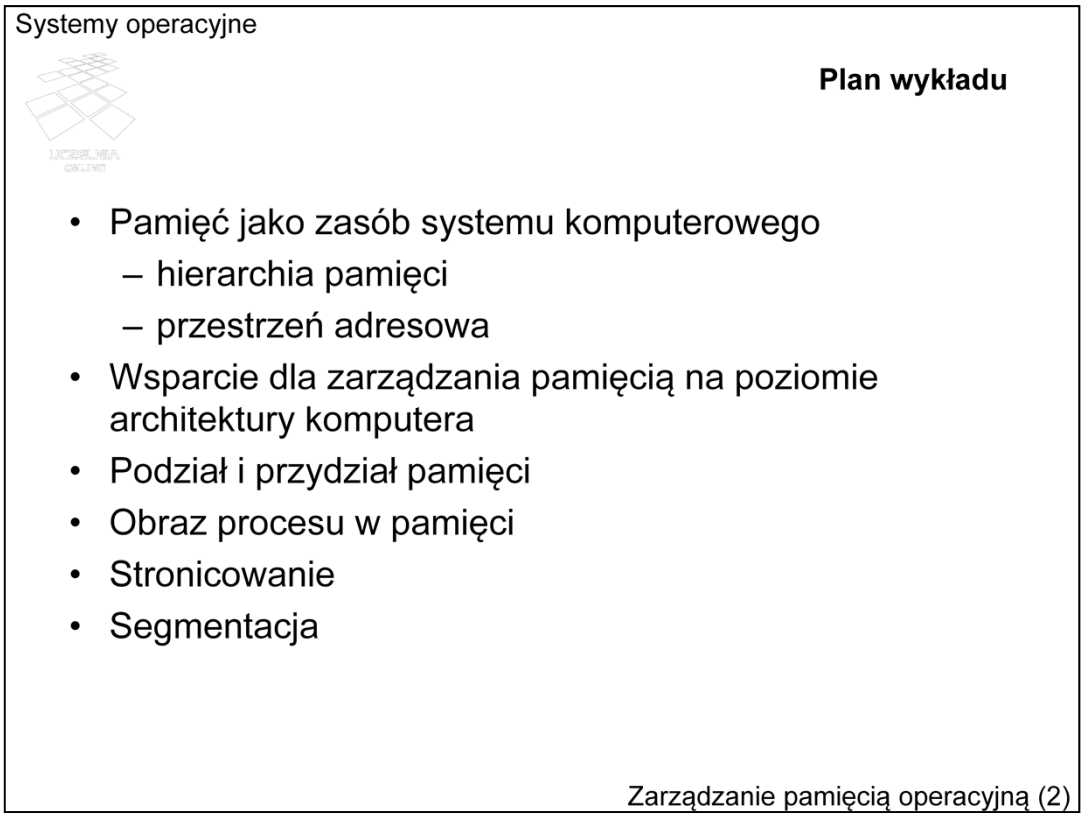
Zarządzanie pamięcią operacyjną

Wykład prowadzą:
Jerzy Brzeziński
Dariusz Wawrzyniak



UCZELNIA
ONLINE

Celem wykładu jest przedstawienie podejść do zarządzania jednym z kluczowych zasobów systemu komputerowego — pamięcią operacyjną. Ponieważ zarządzanie pamięcią operacyjną uwarunkowane jest rozwiązaniami na poziomie architektury komputera, wyodrębnione są zadania realizowane sprzętowo oraz zadania systemu operacyjnego, zmierzające do wykorzystania możliwości sprzętowych.



Wykład rozpoczyna się od przedstawienia podstawowych pojęć, związanych z zarządzaniem pamięcią. Następnie wskazana jest rola układów sprzętowych na poziomie architektury komputera w zarządzaniu pamięcią. Dalsza część dotyczy najważniejszego zadania w zarządzaniu pamięcią, realizowanego przez system operacyjny — przydziału, który ściśle wiąże się z podziałem pamięci. Pozostała część wykładu dotyczy zjawisk wewnątrz przydzielonych obszarów pamięci, czyli tworzenia obrazu procesu, jego ochrony oraz współdzielenia. Na końcu omawiane są techniki odwzorowania logicznej przestrzeni adresowej w fizyczną, wspomagające przy tym zarządzanie pamięcią, czyli stronicowanie i segmentacja.

Systemy operacyjne

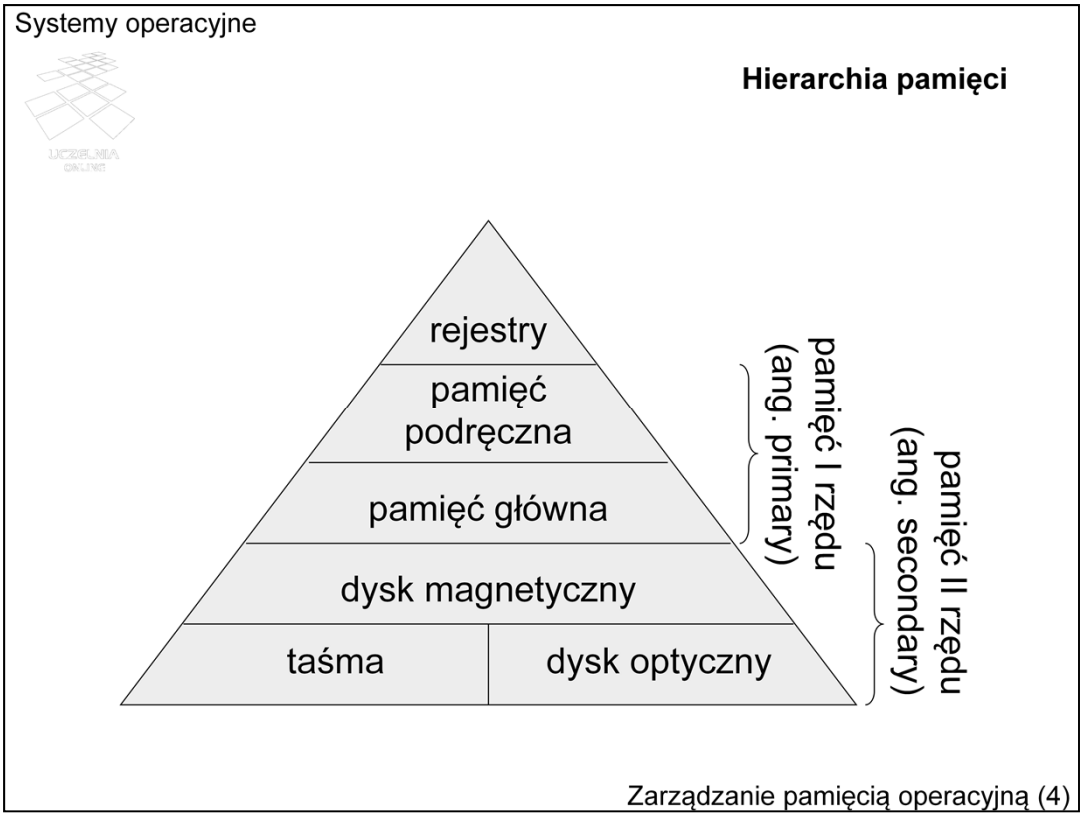
Pamięć jako zasób systemu komputerowego

- Pamięć jest zasobem służącym do przechowywania danych i programów.
- Z punktu widzenia systemu pamięć jest zasobem o strukturze hierarchicznej (począwszy od rejestrów procesora, przez pamięć podręczną, pamięć główną, skończywszy na pamięci masowej), w której na wyższym poziomie przechowywane są dane, stanowiące fragment zawartości poziomu niższego.
- Z punktu widzenia procesu (również procesora) pamięć jest zbiorem bajtów identyfikowanych przez adresy, czyli tablicą bajtów, w której adresy są indeksami.

Zarządzanie pamięcią operacyjną (3)

Pamięć jest kluczowym (obok procesora) zasobem systemu komputerowego dla wykonywania programów. Zarządzanie pamięcią jest jednak dość skomplikowane, gdyż jest ona (poszczególne jej części) jednocześnie wykorzystywana przez wiele procesów często różnych użytkowników oraz przez jądro systemu operacyjnego. Stabilność pracy systemu komputerowego wymaga zatem odpowiedniej ochrony przestrzeni użytkowników, a tym bardziej jądra systemu.

Podstawowe zadania, realizowane w ramach zarządzania pamięcią obejmują przydział pamięci i jej odzyskiwanie, ochronę, udostępnianie w celu współdzielenia, transformację adresów oraz transfer danych pomiędzy poszczególnymi poziomami w hierarchii pamięci. Zadania te podzielone są pomiędzy układy sprzętowe na poziomie architektury komputera, a system operacyjny. Ze względu na efektywność realizacji, zadania takie jak ochrona, transformacja i w dużej części transfer danych realizowane są przez odpowiednie układy sprzętowe. Zadanie systemu operacyjnego sprowadza się do dostarczenia odpowiednich danych tym układom. Dane te wynikają z wcześniejszych decyzji o przydziale pamięci, co już należy do kompetencji systemu operacyjnego.

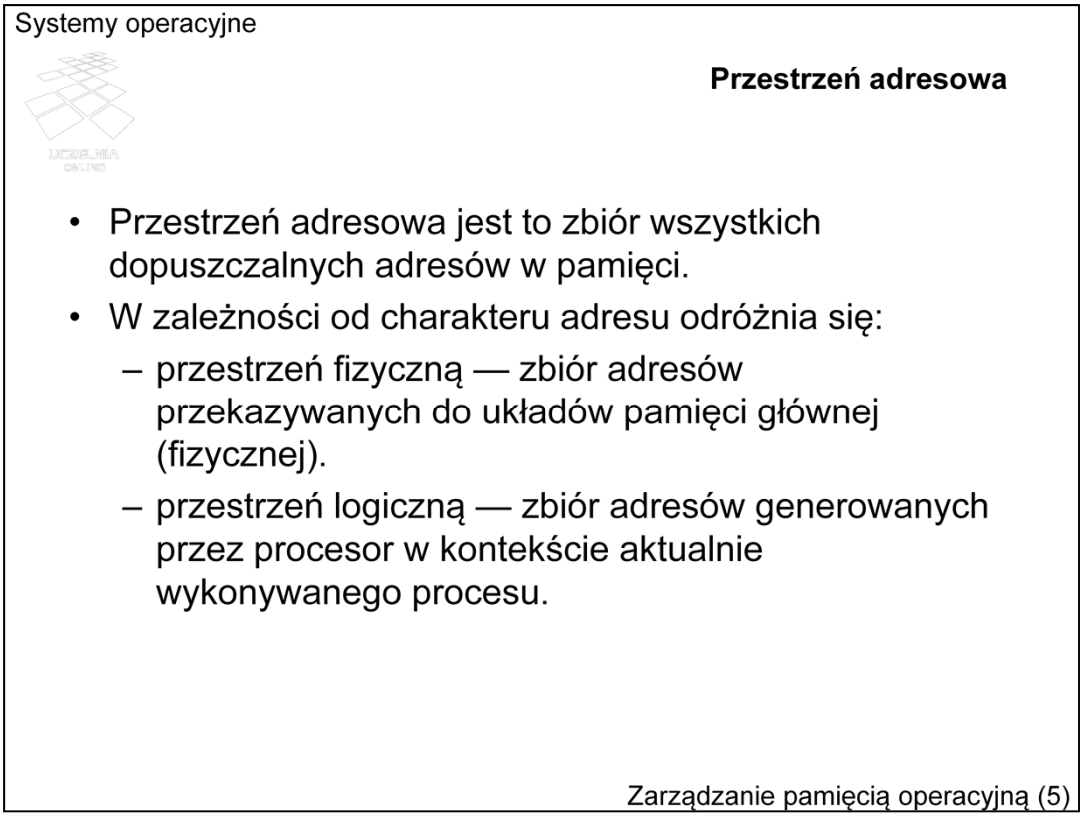


Dodatkowym elementem komplikującym zarządzanie pamięcią jest jej złożona struktura — począwszy od rejestrów procesora, poprzez pamięć podręczną i główną, a skończywszy na pamięci masowej. W hierarchii pamięci na wyższym poziomie znajdują się szybkie układy o niewielkiej pojemności, a w miarę schodzenia niżej zmniejsza się szybkość, a zwiększa pojemność.

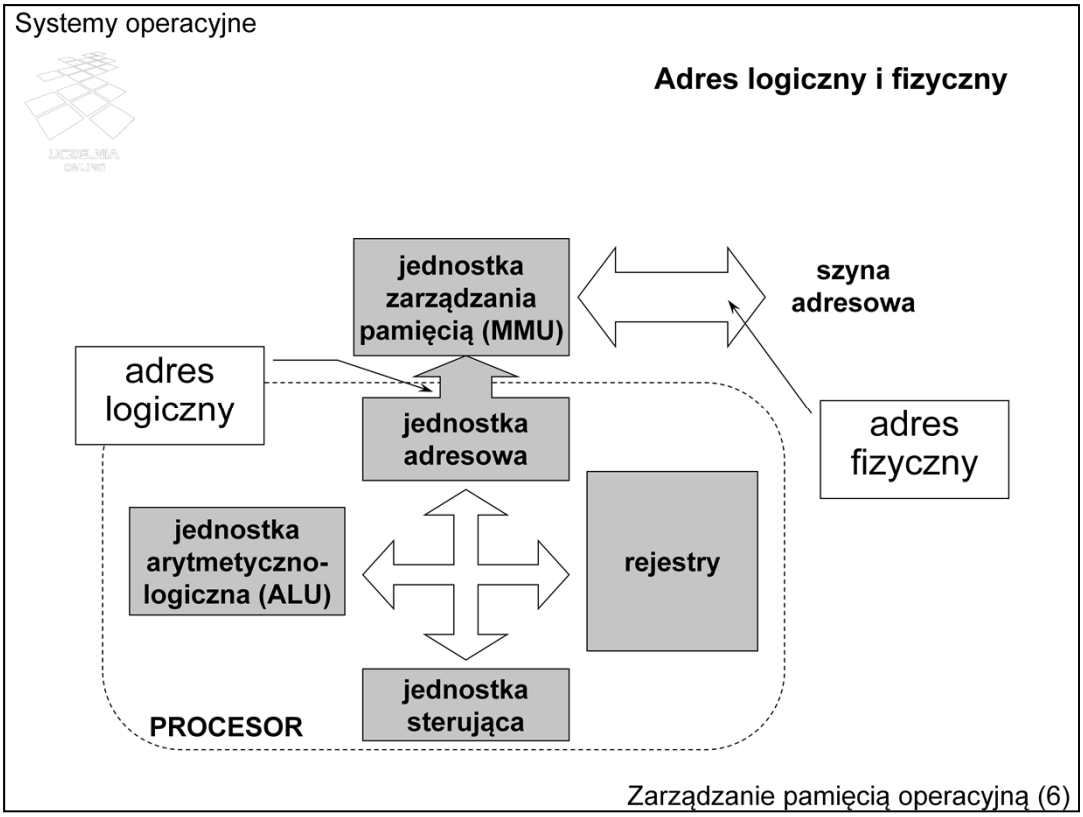
Operowanie zawartością pamięci w takiej złożonej, hierarchicznej strukturze oparte jest na tzw. *zasadzie okna*, zgodnie z którą dane na wyższym (szybszym, ale mniej pojemnym) poziomie stanowią fragment danych, przechowywanych na niższym poziomie.

Zależnie od architektury, procesor adresuje w pamięci bajty, słowa, podwójne słowa itd., a zatem jednostki stosunkowo niewielkie. Takie jednostki obowiązują w transferze pomiędzy rejestrami procesora a pamięcią podręczną lub główną. Pomiędzy niższymi poziomami w hierarchii pamięci transferowane są większe jednostki:

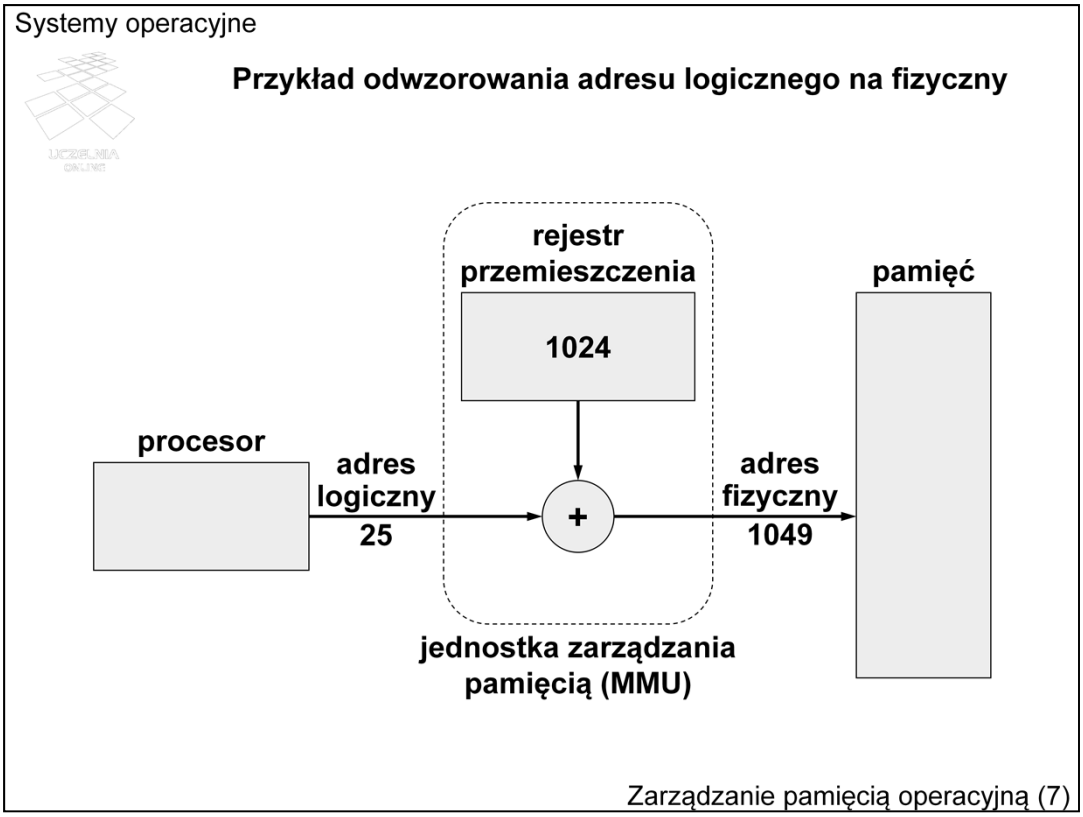
- linijki rzędu od kilkuset bajtów od kilku kilobajtów pomiędzy pamięcią główną a pamięcią podręczną oraz poszczególnymi poziomami samej pamięci podręcznej,
- bloki (sektory lub ich wielokrotności) rzędu od kilku kilobajtów do kilkudziesięciu kilobajtów pomiędzy pamięcią zewnętrzną, a pamięcią główną.



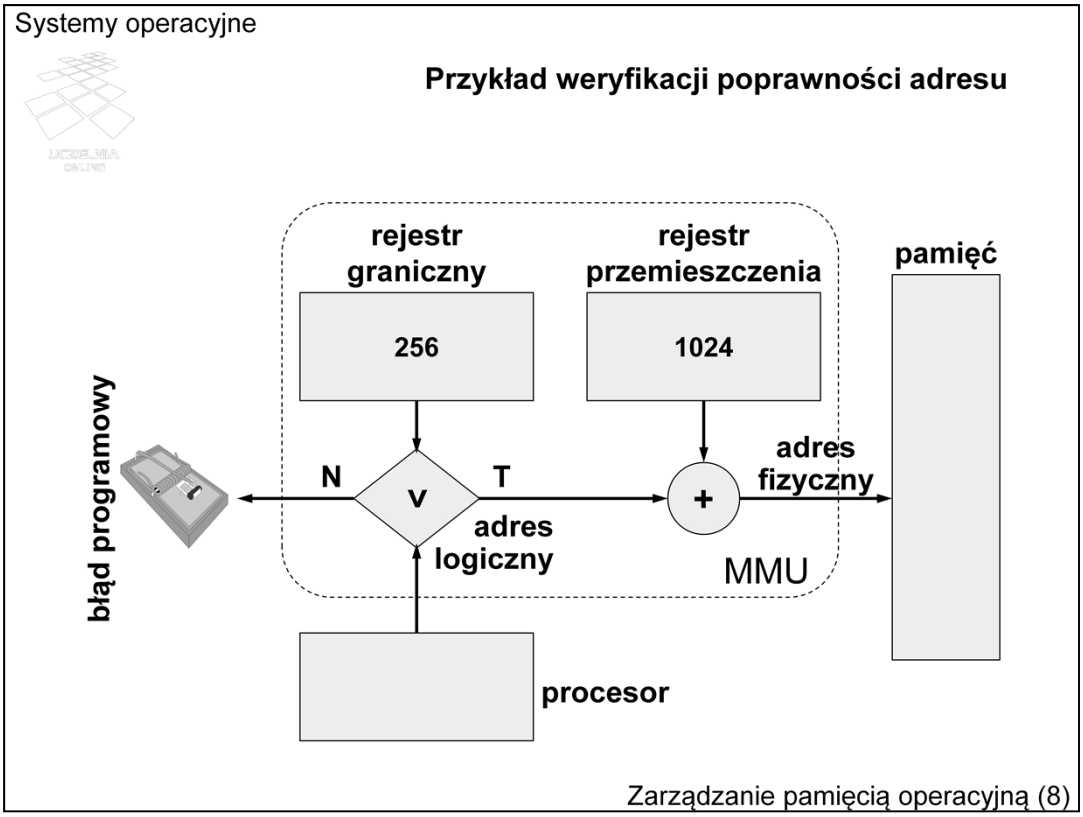
Pamięć postrzegana jest najczęściej jako tablica bajtów, indeksowana przez adresy. Taki jest obraz pamięci zarówno na poziomie architektury komputera, jak i na poziomie systemu operacyjnego, czy też procesu działającego w systemie. Poszczególne komórki pamięci mogą być jednak inaczej identyfikowane na poziomie architektury, a inaczej na poziomie systemu operacyjnego. Te same komórki pamięci mogą być nawet różnie identyfikowane w poszczególnych procesach. Adresy, które identyfikują poszczególne komórki pamięci na poziomie architektury komputera, tworzą fizyczną przestrzeń adresową. Adresy fizyczne przekazywane są szyną adresową magistrali systemowej do układów elektronicznych pamięci. W szczególnym przypadku takimi samymi adresami można się posługiwać na poziomie systemu operacyjnego, ale takie podejście wprowadza sporo ograniczeń, szczególnie uciążliwych w konstrukcji systemów wielozadaniowych. Rozróżnianie przestrzeni fizycznej i logicznej oznacza, że w kontekście procesu komórka pamięci jest inaczej identyfikowana, niż to wynika z jej fizycznego adresu, co wymaga odpowiedniego przekształcenia adres logicznego na fizyczny, zwanego *transformacją adresu*. Za transformację odpowiada układ ściśle współpracujący z procesorem — *jednostka zarządzania pamięcią* (ang. memory management unit — MMU).



Adres logiczny służy do identyfikacji komórek pamięci również na poziomie maszynowym procesora, pracującego w kontekście konkretnego procesu. Adresy przechowywane w rejestrach, wykorzystywanych w różnych trybach adresowania (np. rejestrowym pośrednim, bazowym, indeksowym itp.), czy w liczniku programu (zwanym też wskaźnikiem instrukcji), są adresami **logicznymi**. Procesor rozumiany jest tu jako jednostka funkcjonalna, odpowiedzialna za przetwarzanie. W tym sensie oddzielony jest on od jednostki zarządzania pamięcią, chociaż we współczesnych rozwiązaniach jest z nią strukturalnie zintegrowany. Elementem kontekstu procesu jest zatem również stan jednostki zarządzania pamięcią.



Odwzorowanie adresu logicznego na fizyczny w najprostszym przypadku polega na dodaniu do adresu logicznego, wystawionego przez procesor, pewnej wartości, przechowywanej w rejestrze przemieszczenia w jednostce zarządzania pamięcią. Zawartość komórki pamięci, która w logicznym obrazie procesu zlokalizowana jest pod adresem 25, znajduje się w pamięci fizycznej pod adresem 1049. W ten sposób logiczny obraz procesu można skonstruować, abstrahując od jego fizycznej lokalizacji w pamięci, a przemieszczenie ustalać dopiero w czasie ładowania programu do pamięci lub podczas wykonania.



W systemie wielozadaniowym występuje konieczność ochrony przed zamierzoną lub przypadkową ingerencją jednego procesu w obszar innego procesu lub w obszar jądra systemu operacyjnego. Ochrona jądra systemu operacyjnego wskazana jest również w systemach jednozadaniowych, nie jest jednak elementem krytycznym, gdyż całość zasobów systemu przeznaczona jest na potrzeby jednego przetwarzania. Brak ochrony spowodować może jednak utratę kontroli nad systemem komputerowym w przypadku błędów w programie.

Ochrona pamięci wymaga weryfikacji adresów generowanych przez proces przy każdorazowym odniesieniu do pamięci. W celu weryfikacji adresów w kontekście danego procesu muszą być przechowywane informacje na temat dostępności obszarów pamięci (zakres adresów, tryb dostępu).

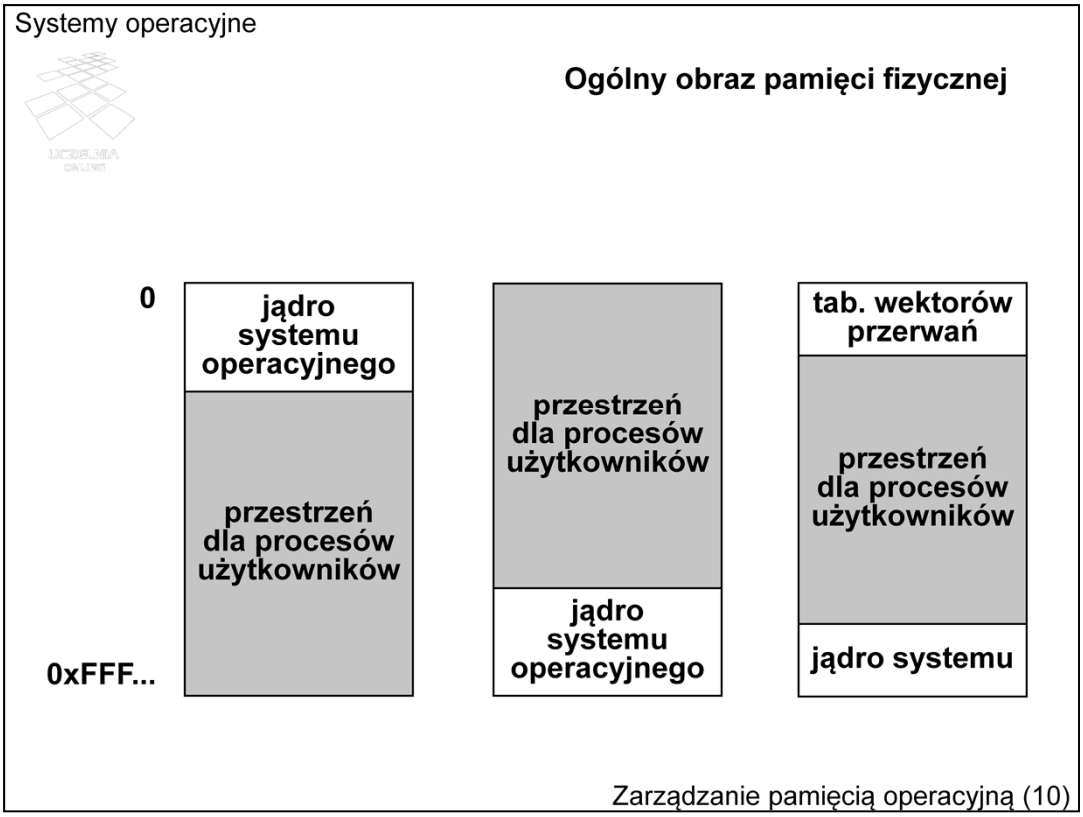
Przykład przedstawia mechanizm transformacji uzupełniony o weryfikację poprawności adresu. Uwzględniając fakt, że procesor wystawia adres logiczny, którego naturalnym dolnym ograniczeniem jest 0, wystarczy sprawdzić, czy adres ten nie wykracza poza górny limit, zgodnie z wielkością przydzielonego obszaru. Jeśli adres logiczny jest mniejszy od wartości granicznej jest poprawny i poddawany jest transformacji. W przeciwnym przypadku następuje zgłoszenie przerwania diagnostycznego.

Na poprzednich slajdach scharakteryzowano krótko najważniejsze zadania, które w zarządzaniu pamięcią realizowane są na poziomie architektury komputera. Zadania te realizowane są przez jednostkę zarządzania pamięcią, pod warunkiem, że udostępnione są odpowiednie dane do transformacji i weryfikacji adresów. W tym miejscu zaczyna się rola systemu operacyjnego.

Zakres adresów fizycznych, dostępnych dla procesu jest konsekwencją przydziału pamięci. Sposób przydziału wiąże się ściśle z podziałem, czyli wyznaczeniem przydzielanych jednostek pamięci lub określeniem zasad ich wyznaczania.

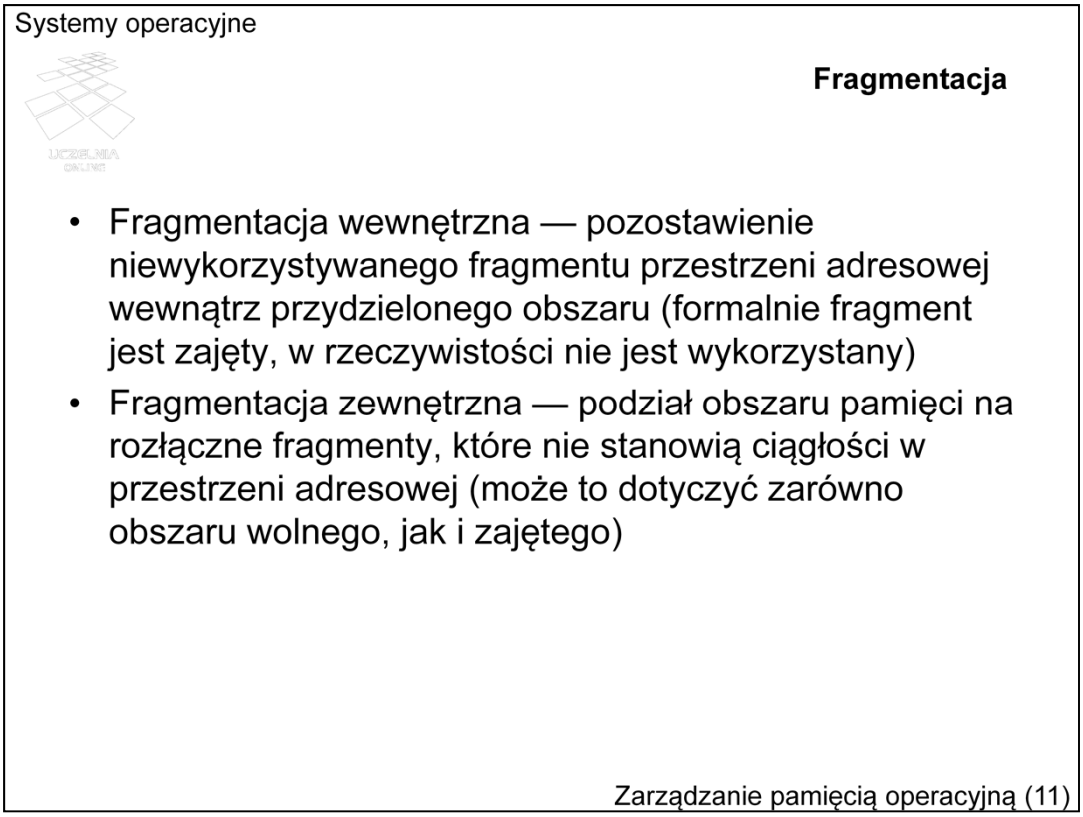
Można wyróżnić:

- podział stały, w którym przydzielane jednostki są odgórnie wyznaczone,
- podział dynamiczny, w którym jednostki definiowane są z pewną dokładnością stosownie do potrzeb,
- podział na bloki bliźniacze, który jest rozwiązaniem pośrednim pomiędzy podziałem statycznym a dynamicznym i polega na połowieniu większych obszarów (zbyt dużych) na dwa mniejsze (o równej wielkości).



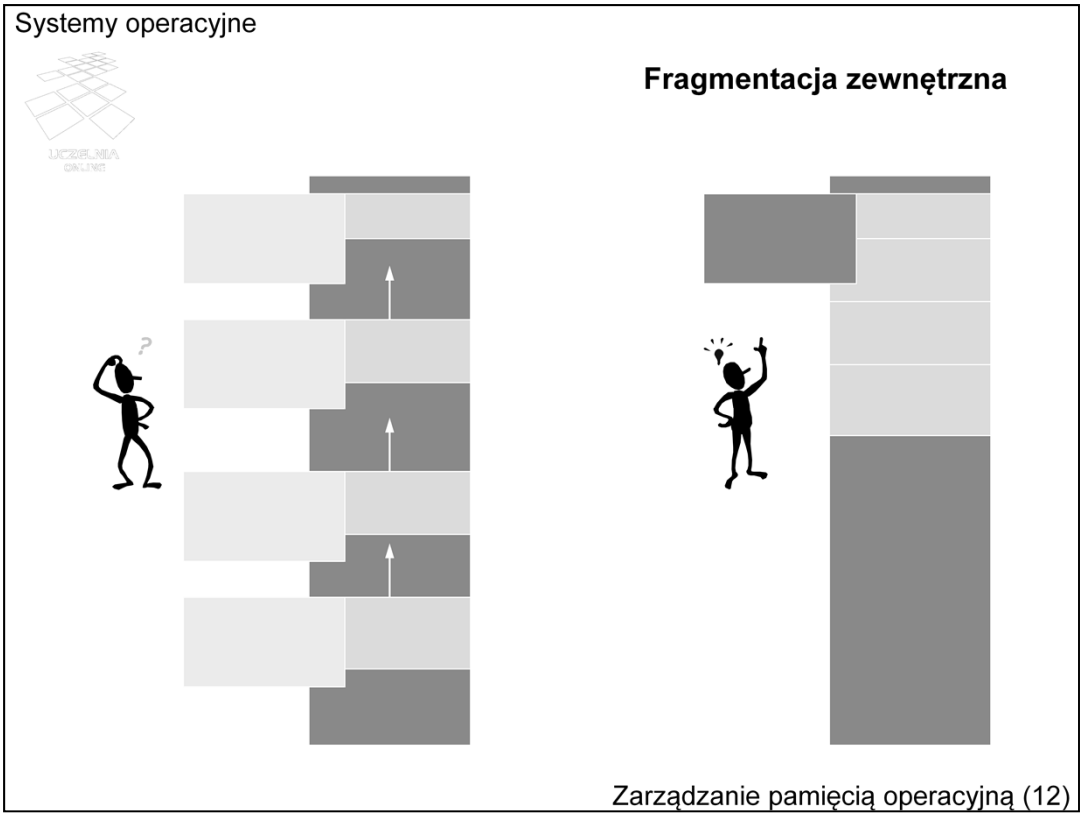
W ogólnym obrazie pamięci fizycznej można wyróżnić część, przeznaczoną na jądro systemu operacyjnego oraz część do dyspozycji procesów użytkownika, którą przydziela oczywiście zarządca pamięci (odpowiedni moduł systemu operacyjnego). Jądro zajmuje najczęściej początkowy lub końcowy obszar pamięci fizycznej, ale właściwa lokalizacja zależy od rozwiązań na poziomie architektury komputera.

Omawiane zagadnienia podziału i przydziału pamięci można odnieść zarówno do przestrzeni, przeznaczonej na procesy użytkownika, jak i do obszaru jądra systemu operacyjnego. Przestrzeń dla procesów użytkownika jest przydzielana w reakcji na żądania użytkowników, związane z tworzeniem procesów lub przydzielaniem im dodatkowej pamięci. Jądro przeznacza pewien obszar przydzielonej pamięci na tymczasowe potrzeby, wynikające z bieżąco realizowanych żądań zasobowych procesów lub obsługiwanych urządzeń.



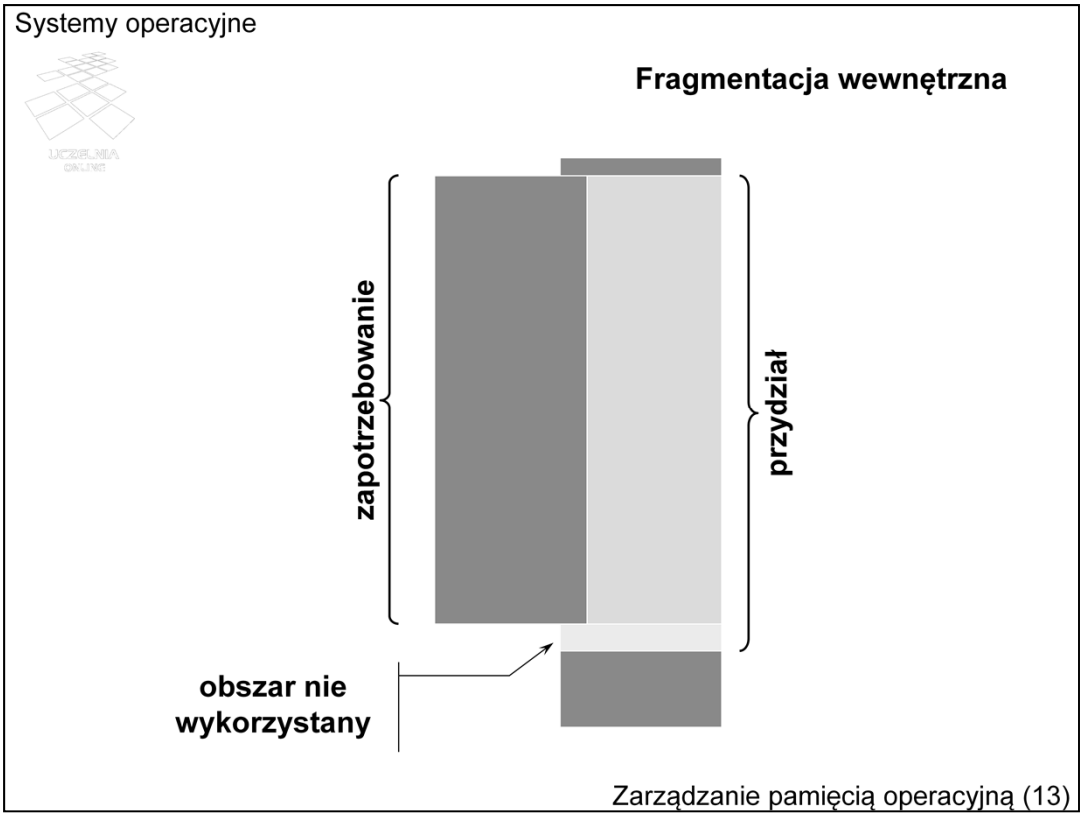
Zjawiskiem związanym z podziałem i przydziałem pamięci jest fragmentacja. Pod pojęciem fragmentacji kryją się dwa osobne zjawiska określane jako *fragmentacja zewnętrzna* i *fragmentacja wewnętrzna*. Fragmentacja wewnętrzna dotyczy niewykorzystania pamięci wewnątrz przydzielonego bloku i najczęściej jest skutkiem operowania przez system większymi jednostkami, niż dokładność specyfikacji potrzeb ze strony aplikacji lub jądra systemu. Ten rodzaj fragmentacji jest charakterystyczny dla systemów z podziałem stałym.

Fragmentacja zewnętrzna oznacza podział na osobne części. Problem fragmentacji zewnętrznej ujawnia się najczęściej w zarządzaniu wolną przestrzenią. W niektórych podejściach do zarządzania pamięcią (np. w stronicowaniu) można też go odnieść do obszarów pamięci przydzielonych procesom. Pofragmentowanie wolnej przestrzeni bierze się stąd, że przydzielane są kolejne fragmenty pamięci, a następnie część z nich jest zwalniana, a część pozostaje dalej zajęta.



W przedstawionym obrazie pamięci wolna przestrzeń jest podzielona na 4 fragmenty, z których żaden nie jest wystarczająco duży, żeby pomieścić w ciągłym obszarze pamięci blok danych na potrzeby procesu. Gdyby jednak udało się scalić wolne fragmenty w jeden duży obszar, wystarczyłoby miejsca na więcej niż 2 takie bloki.

Możliwość scalenia zależy od sposobu wiązania adresów. Sensowna realizacja scalenia wolnych fragmentów wymaga ustalania adresów fizycznych w czasie wykonania przy wsparciu jednostki zarządzania pamięcią.



Fragmentacja wewnętrzna wynika najczęściej z ograniczeń na rozmiar przydzielanej jednostki. Nie jest to jednak jedyny przypadek. W przedstawionym przykładzie przydział dokładnie tylu bajtów, ile wynosi zapotrzebowanie, powoduje, że koszt utrzymania bardzo małego obszaru wolnego jest niewspółmiernie duży, np.:

- mogłoby się okazać, że dane o obszarze zajmują więcej bajtów, niż rozmiar tego obszaru (co mogłoby być nawet przyczyną pewnych problemów implementacyjnych),
- wszystkie algorytmy przydziału uwzględniałyby ten obszar podczas wyszukiwania wolnego miejsca, podczas gdy prawdopodobieństwo jego wykorzystania byłoby niewielkie.

Dlatego wolny obszar przydzielany jest w całości, ale nie jest w pełni wykorzystany. Powstaje wiec fragmentacja wewnętrzna, wynikająca z decyzji zarządcy, a nie z konfiguracji systemu.

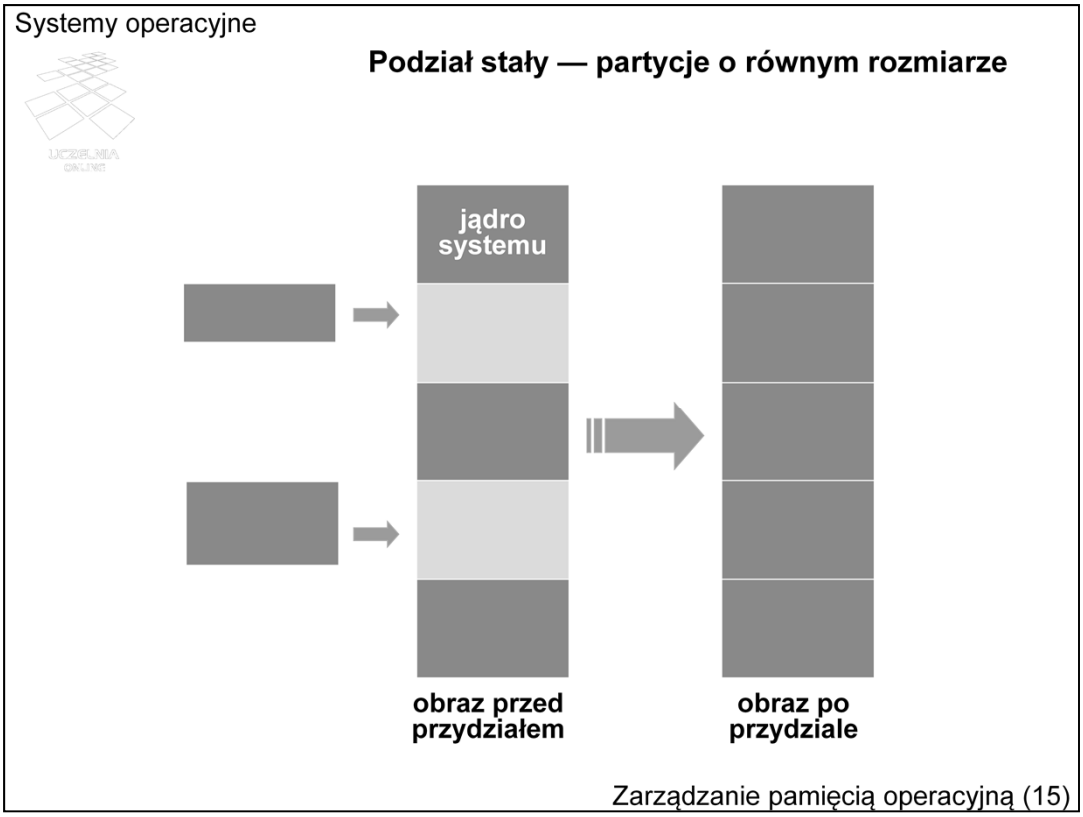
Systemy operacyjne

Podział stały

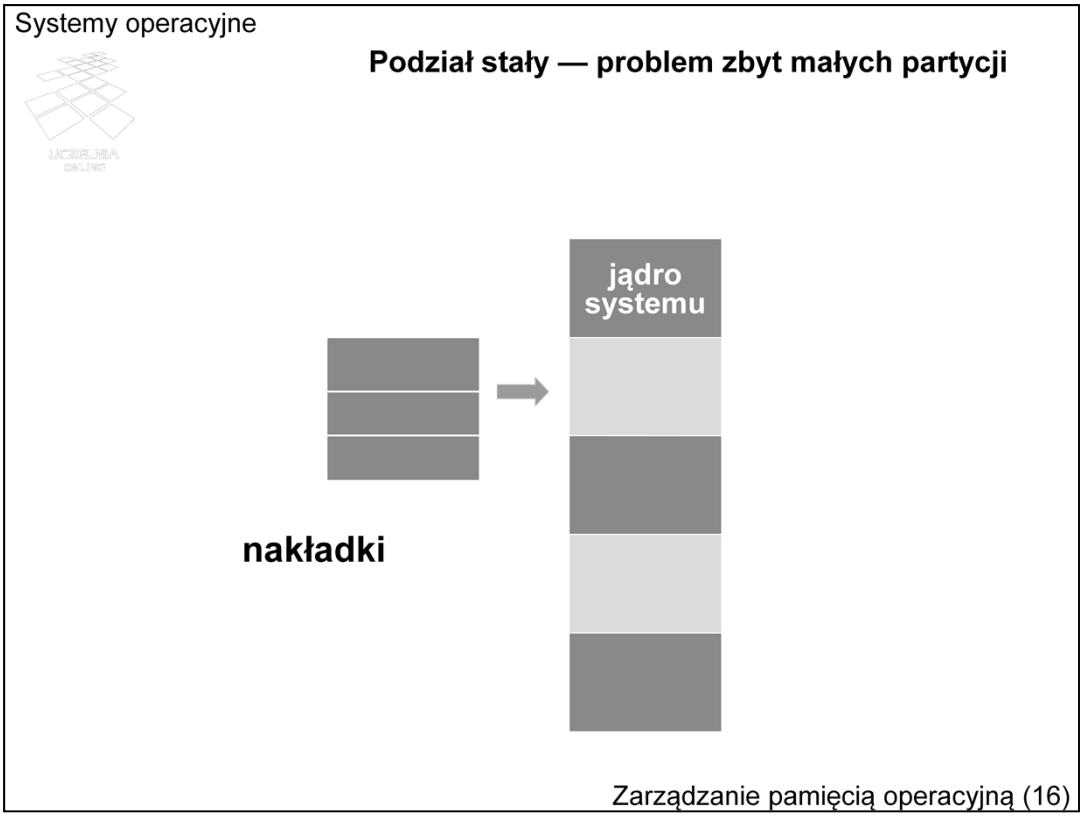
- Podział pamięci na stałe obszary (strefy, partycje), których rozmiar i położenie ustalane są na etapie konfiguracji systemu.
- Przydział całego obszaru o rozmiarze większym lub równym zapotrzebowaniu, określonym w żądaniu.
- Zalety: łatwość implementacji i zarządzania
- Wady: słaba efektywność wykorzystania pamięci (fragmentacja wewnętrzna, ograniczona odgórnie liczba jednocześnie przydzielonych partycji).

Zarządzanie pamięcią operacyjną (14)

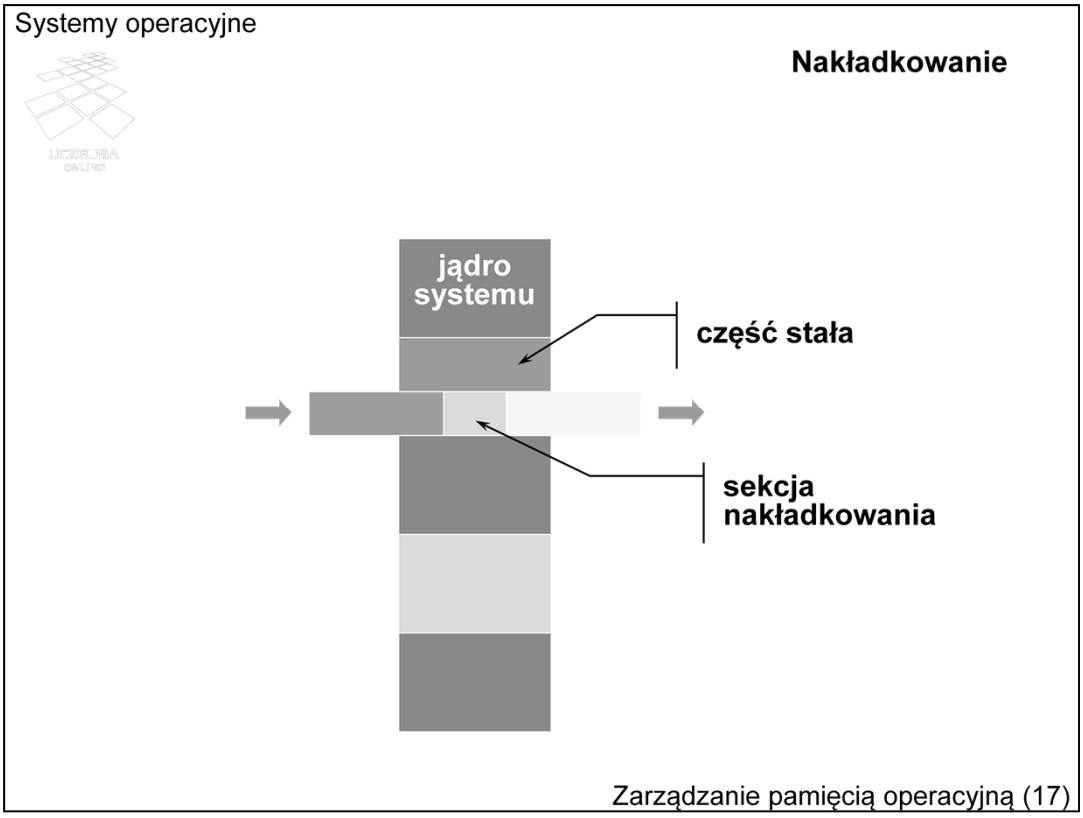
W podejściu z podziałem stałym nie można oczekiwać przydziału ciągłego obszaru pamięci o rozmiarze większym, niż to wynika z podziału, dokonanego na etapie konfiguracji systemu. Konsekwencją podziału stałego przestrzeni procesów użytkownika jest konieczność ograniczenia rozmiaru procesu do rozmiaru partycji. Z drugiej strony, mniejszym procesom również przydzielane są obszary pamięci, wynikające z takiego podziału, co oznacza marnowanie pamięci w związku z fragmentacją wewnętrzną.



W najprostszym przypadku partycje mają ten sam rozmiar. Realizacja żądania polega zatem na znalezieniu jakiejkolwiek wolnej partycji. W przedstawionym przykładzie realizacji żądań przydziału pamięci dla dwóch procesów obie wolne partycje zostają zajęte w całości, pomimo że potrzeby procesów są mniejsze.



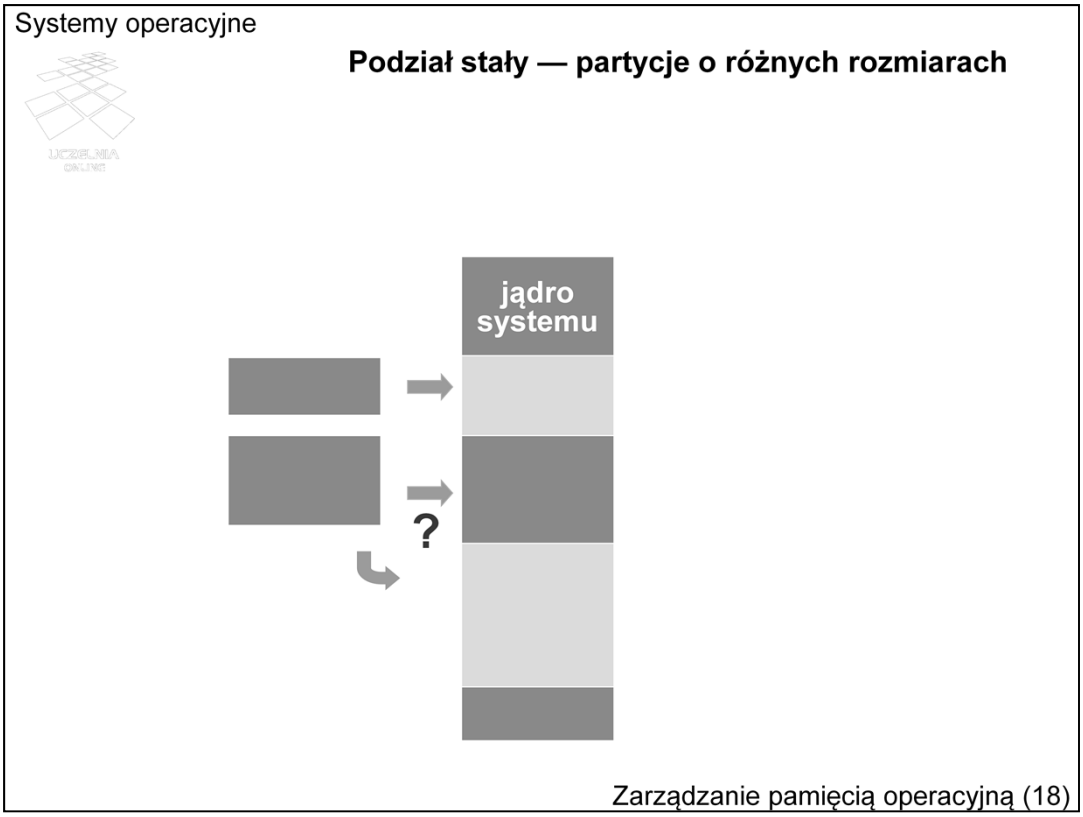
Istotnym problemem może być przypadek zażądania przydziału obszaru większego, niż udostępniony przez system w wyniku podziału. W przypadku przydziału pamięci na dane (np. w jądrze) ograniczenia, wynikające z podziału stałego, można przewidzieć wcześniej i dostosować odpowiednio struktury danych. W przypadku programu można zastosować technikę, zwaną nakładkowaniem. Nakładkowanie polega na podziale **kodu** (nie danych) na części niezależne od siebie i wymianie w miarę potrzeb jednej części — nakładki (ang. overlay) — na inną.



W programie wydziela się część stałą, która zawsze znajduje się w pamięci oraz nakładki, które wiąże się z tzw. sekcją nakładkowania. Na każdą sekcję wydzielona jest pewna część pamięci. Nakładki w ramach tej samej sekcji podlegają wymianie — jedna nakładka usuwa inną. W programie może być kilka takich sekcji.

Z nakładkami wiążą się pewne ograniczenia. Stan nakładki usuwanej z pamięci nie jest nigdzie zapisywany, w związku z tym nie może ona ulegać zmianie — może zawierać tylko kod, ewentualnie stałe. Ograniczone są też możliwości przekazywania danych i sterowania pomiędzy nakładkami w tej samej sekcji, zatem nie można się odwołać z jednej nakładki do drugiej w tej samej sekcji. Przekazywanie danych i sterowania odbywa się najczęściej za pośrednictwem części stałej, czyli w tej części (ewentualnie w nakładce w innej sekcji) znajdują się wywołania podprogramów zlokalizowanych w nakładkach. Podział na nakładki wymaga dokładnej znajomości kodu i przepływu sterowania. Ponieważ nakładkowanie stosowane jest w przypadku dużych programów, podział na nakładki jest złożonym zadaniem. Może też być źródłem dodatkowych, trudno wykrywalnych błędów.

Nakładkowanie nie wymaga wsparcia ze strony jądra systemu operacyjnego, co najwyżej ze strony pewnych narzędzi do tworzenia kodu. Ze względu na uciążliwość technika ta stosowana jest jednak tylko w przypadku istotnych ograniczeń na rozmiar pamięci fizycznej w stosunkowo prostych architekturach. Zalecanym rozwiązaniem, wymagającym jednak wsparcia zarówno na poziomie architektury, jak i na poziomie systemu operacyjnego, jest pamięć wirtualna, która zostanie omówiona w następnym module.



Odpowiedzią na zróżnicowane pod względem wielkości żądania może być podział na partycje o różnych rozmiarach. Realizacja żądania wymaga znalezienie partycji, odpowiednio dobrze dopasowanej do wielkości zapotrzebowania. Może jednak powstać problem decyzyjny w przypadku, gdy najlepiej dopasowane partycje są zajęte, a wolne są partycje większe. Optymalizując przepustowość można przydzielić partycję większą (kosztem fragmentacji wewnętrznej), a optymalizując wykorzystanie pamięci należałoby wstrzymać przydział do momentu zwolnienia najlepiej dopasowanej partycji.

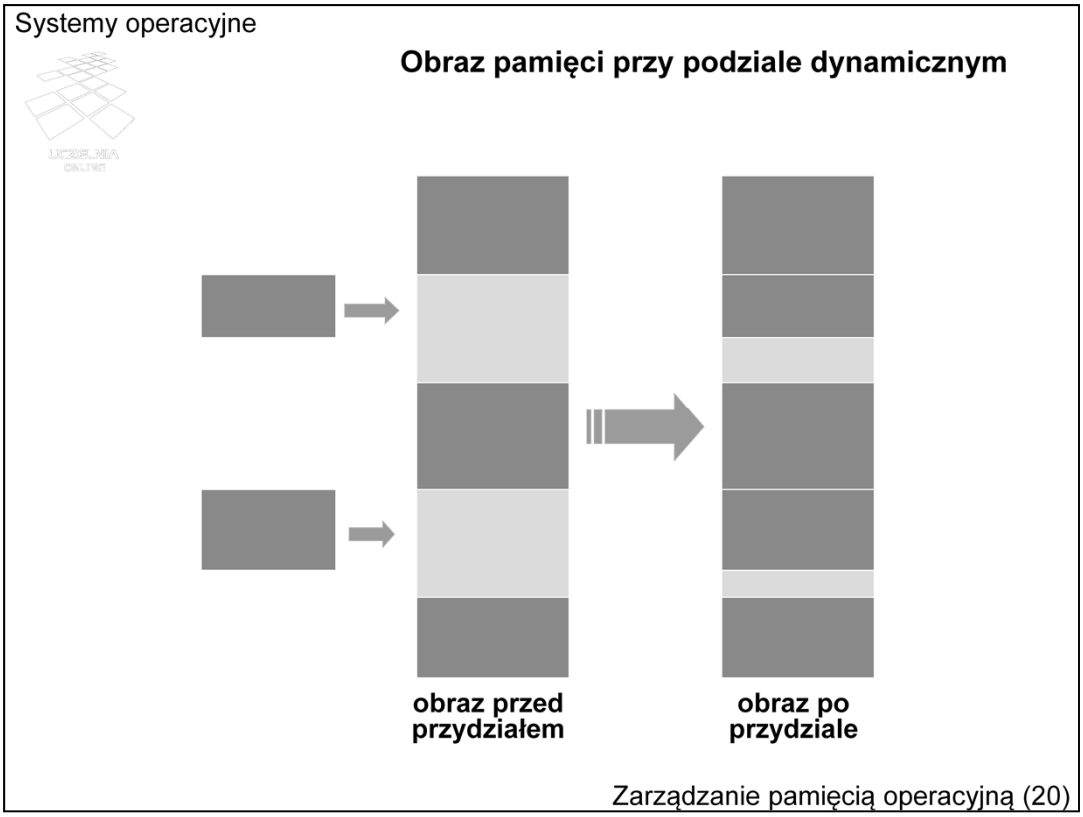
Systemy operacyjne

Podział dynamiczny

- Podział pamięci tworzony jest w czasie pracy systemu stosownie do żądań procesów.
- Proces ładowany jest w obszar o rozmiarze dosyć dokładnie odpowiadającym jego wymaganiom.
- Zalety: lepsze wykorzystanie pamięci (brak fragmentacji wewnętrznej)
- Wady: skomplikowane zarządzanie, wynikające z konieczności utrzymywania odpowiednich struktur danych w celu identyfikacji obszarów zajętych oraz wolnych.

Zarządzanie pamięcią operacyjną (19)

Istotą podziału dynamicznego jest operowanie stosunkowo niewielkimi jednostkami i przydzielanie bloków, stanowiących ciąg kolejnych jednostek w stosownej do zapotrzebowania liczbie. Przydzielany jest zatem ciągły obszar pamięci z dokładnością do rozmiaru jednostki. Jednostka może obejmować 1 bajt, ale częściej operuje się nieco większymi jednostkami, np. 16 bajtów (tzw. paragraf). W przypadku większych jednostek możliwa jest niewielka fragmentacja wewnętrzna.



W przypadku podziału dynamicznego po przydzieleniu odpowiedniego bloku reszta obszaru pozostaje wolna i jest do dyspozycji na potrzeby kolejnych żądań.

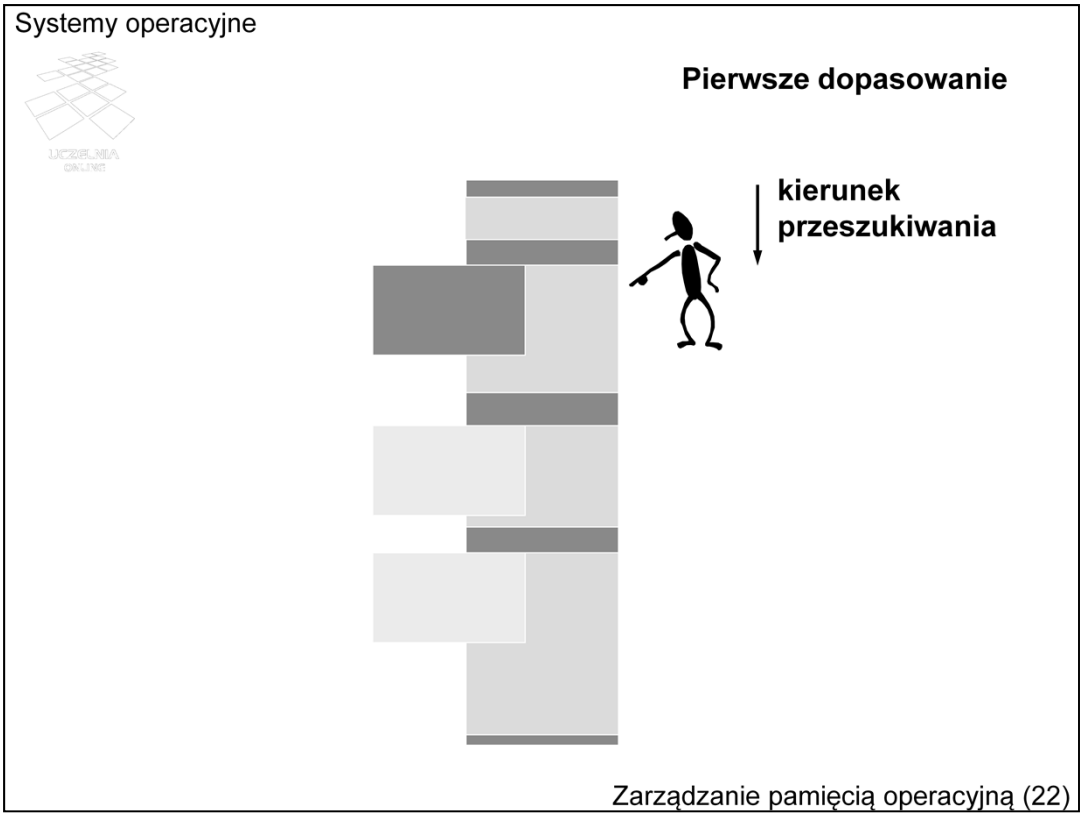
Systemy operacyjne

Podział dynamiczny — problem wyboru bloku

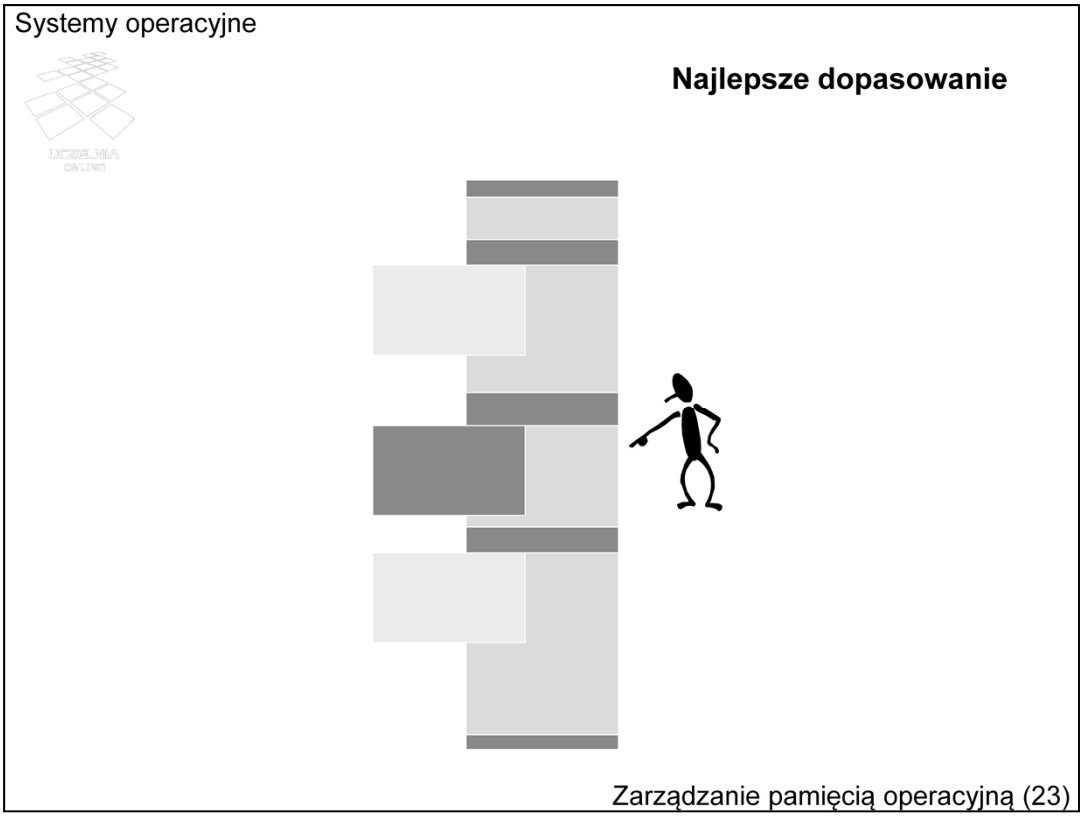
- Pierwsze dopasowanie (ang. first fit) — przydziela się **pierwszy** wolny obszar (tzw. dziurę) o wystarczającej wielkości. Poszukiwanie kończy się po znalezieniu takiego obszaru.
- Najlepsze dopasowanie (ang. best fit) — przydziela się **najmniejszy** dostatecznie duży wolny obszar pamięci. Konieczne jest przeszukanie wszystkich dziur.
- Następne dopasowanie — podobnie jak pierwsze dopasowanie, ale poszukiwania rozpoczyna się od miejsca ostatniego przydziału.
- Najgorsze dopasowanie (ang. worst fit) — przydziela się **największy** wolny obszar pamięci. Konieczne jest przeszukanie wszystkich dziur.

Zarządzanie pamięcią operacyjną (21)

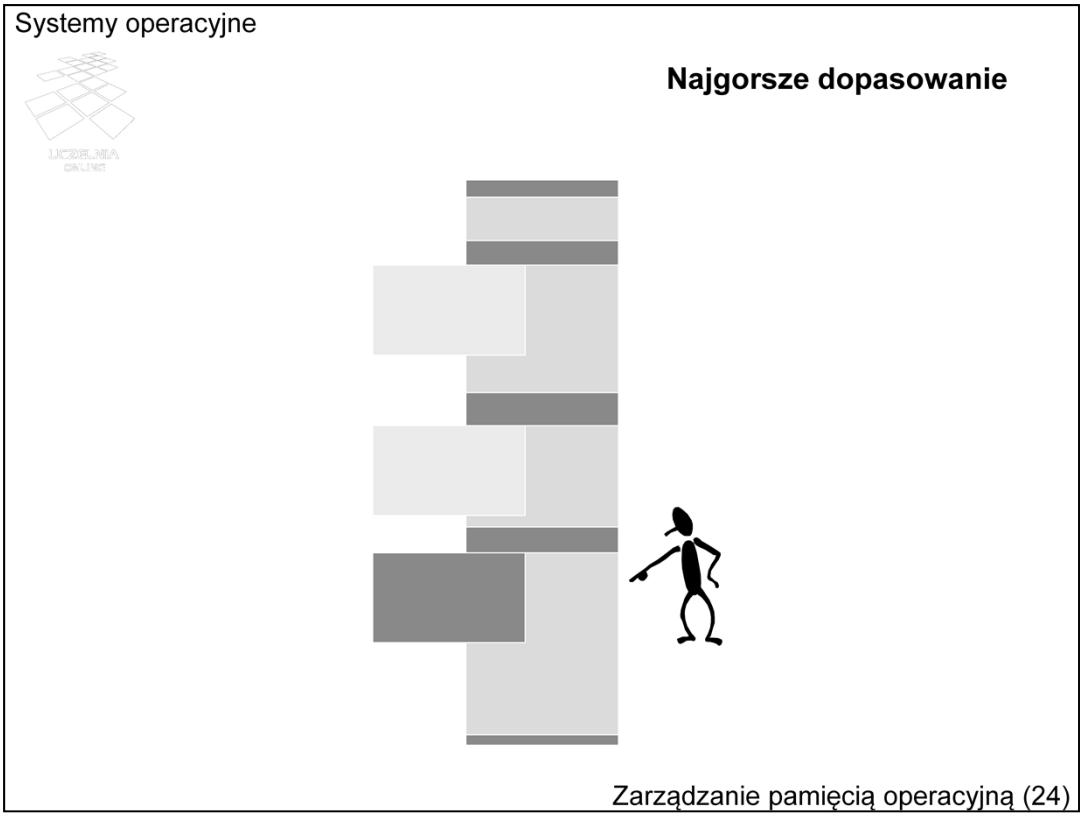
Wolne obszary mogą mieć różne rozmiary i trudno z góry ustalić, gdzie znajduje się odpowiedni obszar. Wyróżnia się 4 strategie poszukiwania odpowiedniego obszaru, tzw. *dziury*.



Pierwszy wolny blok jest zbyt mały, więc przydzielany jest następny wolny blok. Jest on jednak większy niż zapotrzebowanie, dlatego po przydzieleniu pozostanie jeszcze trochę wolnego miejsca. Jest to metoda szybka, biorąc pod uwagę średni czas wyszukiwania. Warto zwrócić uwagę, że blok dałoby się dopasować również w pozostałe wolne obszary, ale zgodnie z kierunkiem przeszukiwania nie były one w ogóle analizowane.

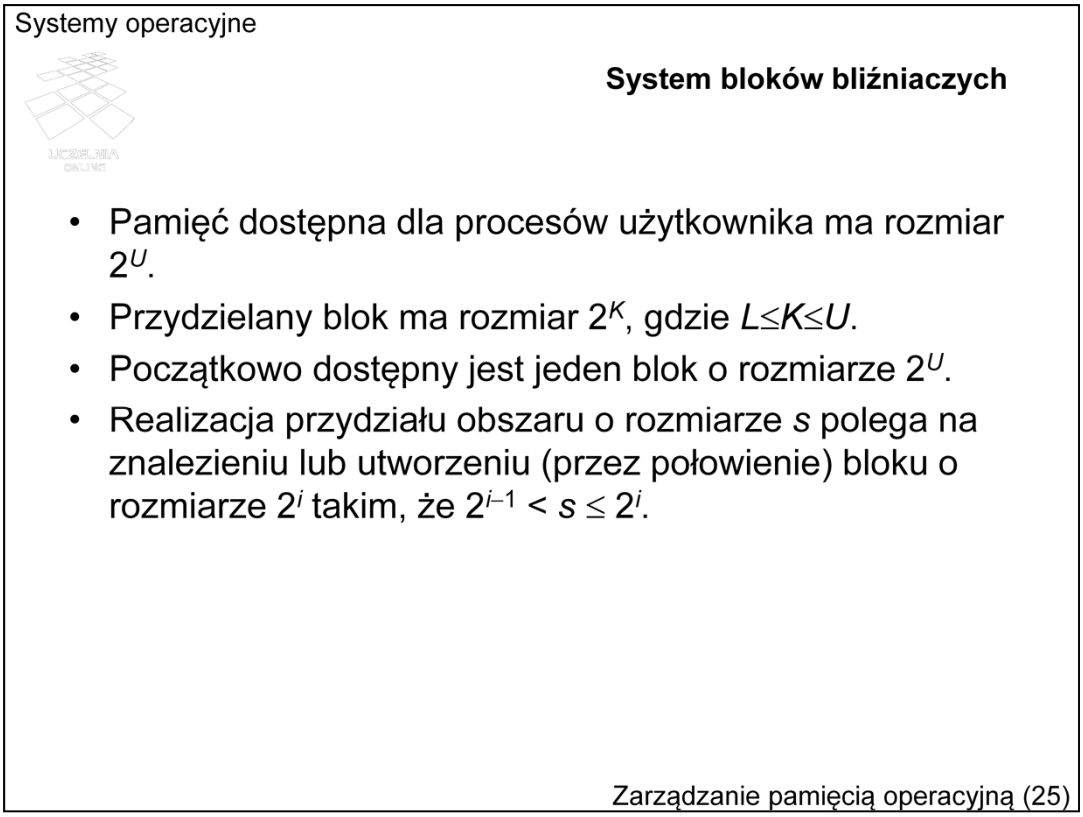


Poszukiwany jest taki obszar wolny, żeby po przydziale pozostało po nim jak najmniej wolnego miejsca. Wymaga to przeszukania wszystkich dziur (dlatego metoda jest stosunkowo powolna), chyba że znajdzie się obszar dokładnie odpowiadający zapotrzebowaniu lub pozostały do przeszukania obszar jest mniejszy niż zapotrzebowanie.



Znalezienie największego wolnego obszaru wymaga również przeszukania wszystkich wolnych dziur, chyba że znajdzie się dziurę większą niż połowa zakresu pamięci, jaki pozostał jeszcze do przeszukania lub obszar pozostały do przeszukania jest niewiększy niż dotychczas znaleziona największa dziura.

Metoda nie jest zbyt często stosowana, jednak jej idea jest taka, żeby pozostawiać stosunkowo duże wolne obszary, gdyż zarządzanie małymi obszarami jest często nieefektywne (o czym wspomniano przy omawianiu fragmentacji wewnętrznej).

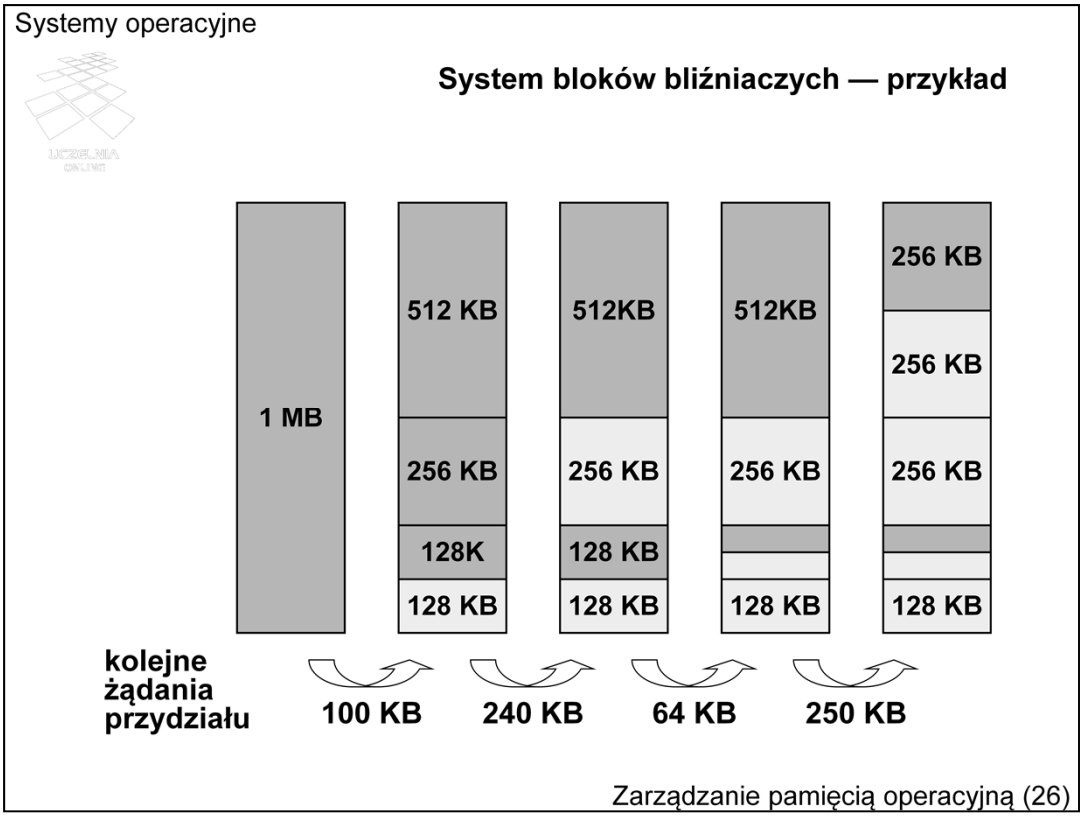


Metoda bloków bliźniaczych (ang. buddy) polega na sukcesywnym dzieleniu dostępnego obszaru pamięci na połowy i przydziale najlepiej dopasowanego bloku, którego rozmiar jest potęgą przy podstawie 2. Jest przy tym ustalony minimalny rozmiar przydzielanego bloku (wykładnik L).

Jest to metoda pośrednia pomiędzy przydziałem stałym a dynamicznym. Rozmiar partycji nie jest ustalony odgórnie, ale możliwość dopasowania go do żądanej wielkości jest ograniczona. Najmniej korzystny przypadek ma miejsce, gdy wielkość żądanego obszaru jest trochę większa niż potęga dwójki, gdyż prawie połowa przydzielonego bloku pozostaje niewykorzystana (fragmentacja wewnętrzna).

Takie podejście ułatwia jednak zarządzanie, gdyż przyspiesza wyszukiwanie odpowiedniego bloku. Wolne bloki o danym rozmiarze identyfikowane są poprzez listę, której czoło znajduje się w tablicy. Indeks tablicy odpowiada wykładnikowi potęgi dwójki, określającej rozmiar bloku.

Metoda bloków bliźniaczych stosowana jest w systemie Linux, przede wszystkim na wewnętrzne potrzeby jądra. Również podczas przydziału stron pamięci dla procesów zarządca stara się je grupować i w przypadku większych żądań przydzielać ciąg stron o kolejnych numerach, stanowiących bliźniaczy blok.



Przykład pokazuje realizację ciągu żądań przydziału pamięci w metodzie bloków bliźniaczych. Do dyspozycji jest obszar 1 MB (=1024 KB). W odpowiedzi na żądanie przydziału 100 KB następuje podział bloku 1 MB na 2 bloki po 512 KB, z których jeden jest dalej dzielony na 2 bloki po 256 KB, a jeden z tych bloków z kolei dzielony jest znowu na 2 bloki po 128 KB. Dalszy podział nie ma już sensu, gdyż 64 KB to za mało w stosunku do zapotrzebowania. Żądanie przydziału 240 KB będzie zaspokojone niemal natychmiast, ponieważ dokonany wcześniej podział wyodrębnił już jeden blok o rozmiarze 256 KB, a jego połowa byłaby zbyt małym obszarem. Kolejne żądanie — przydziału 64 KB — wymaga przepołowienie wydzielonego już bloku 128 KB. Następne żądanie — 250 KB, wymaga bloku 256 KB, który może powstać przez podział bloku 512 KB. Warto zwrócić uwagę, że gdyby ostanie żądanie dotyczyło bloku o rozmiarze trochę większym, np. 260 KB, przydzielony musiałby być cały blok 512 KB.

Systemy operacyjne

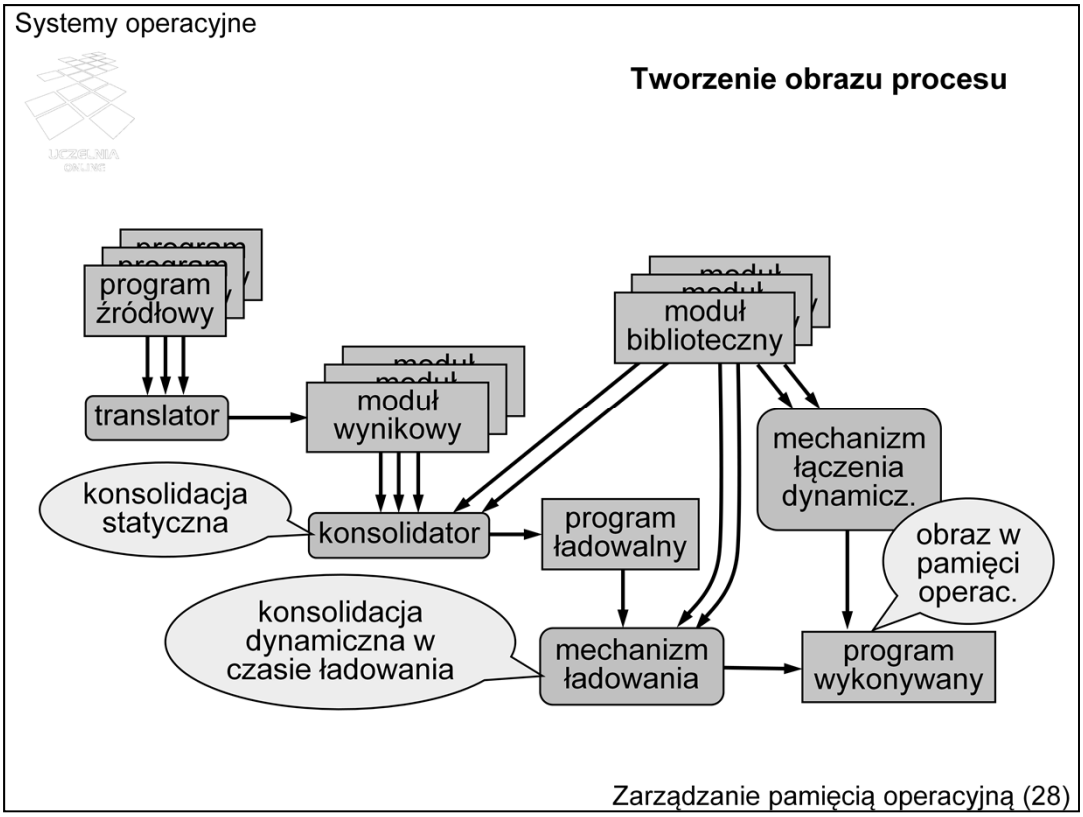
Obraz procesu w pamięci

- Tworzenie obrazu
  - Kompilacja
  - Konsolidacja
  - Ładowanie kodu
- Relokacja
- Ochrona
- Współdzielenie

Zarządzanie pamięcią operacyjną (27)

Dotychczas omawiane zagadnienia dotyczyły zarządzania pamięcią bez interpretowania jej zawartości. Jednak takie zagadnienia, jak ochrona, czy transformacja adresu zostały tylko zasygnalizowane. Zanim nastąpi ich rozwinięcie, omówienia wymagają kwestie, dotyczące obrazu procesu w pamięci. Ponieważ tworzenie obrazu procesu jest zagadnieniem z dziedziny translatorów, poruszone zostaną tylko najważniejsze zagadnienia, które mają wpływ na funkcjonowanie systemu operacyjnego.

Tworzenie obrazu procesu obejmuje kompilację, konsolidację i ładowanie. W zależności od podejścia do wiązania adresów (wynikających często z możliwości sprzętu), można też rozważać relokację procesu w pamięci. Osobnym zagadnieniem, choć już wcześniej wspomnianym jest ochrona. Cześć obrazu (np. kod programu, kod funkcji bibliotecznych, niektóre dane) może być wspólna dla kilku procesów. Takie podejście poprawia efektywność i otwiera drogę do współpracy procesów, ale przy ochronie pamięci wymaga odpowiedniego odseparowania fizycznej i logicznej przestrzeni adresowej.



Tworzenie obrazu procesu zaczyna się od programu źródłowego. Program taki jest kompilowany do przemieszczalnego modułu wynikowego, a następnie łączony (konsolidowany) z modułami bibliotecznymi, do których są odwołania w kodzie. Konsolidację można jednak odłożyć do czasu ładowania, a nawet wykonywania kodu. Konsolidacja przed ładowaniem określana jest jako *statyczna*, a w jej wyniku powstaje moduł absolutny. Konsolidacja w czasie ładowania lub wykonania określana jest jako konsolidacja *dynamiczna*.

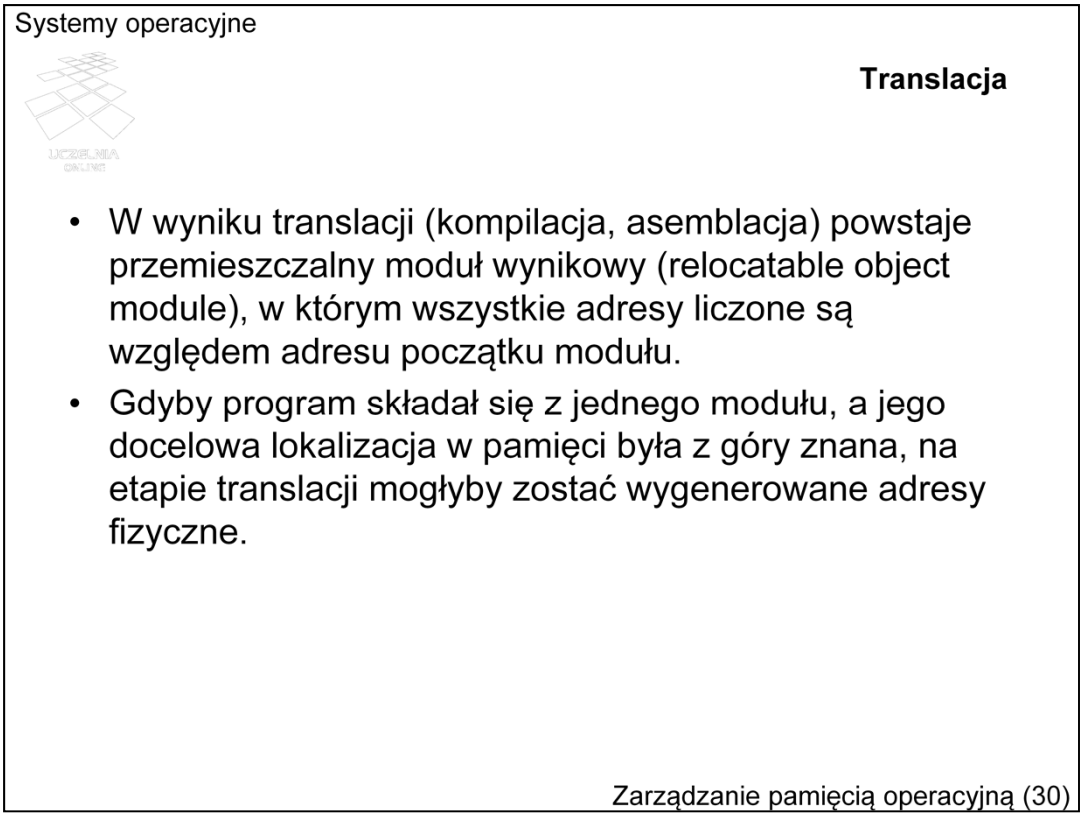
Systemy operacyjne

Wiązanie i przekształcanie adresów

- W modelu von Neumana adresy dotyczą rozkazów (instrukcji) oraz zmiennych (danych).
- Jeśli w programie źródłowym występują adresy, to mają one najczęściej postać symboliczną (etykiety w asemblerze) lub abstrakcyjną (wskaźniki w C lub Pascalu).
- Adresy związane z lokalizacją pojawiają się na etapie translacji i są odpowiednio przekształcane aż do uzyskania adresów fizycznych.

Zarządzanie pamięcią operacyjną (29)

Każdemu etapowi tworzenia obrazu procesu w pamięci towarzyszy odpowiednie przekształcanie adresów, począwszy od etykiet i innych symboli, a skończywszy na fizycznych adresach komórek pamięci.



Translacja oznacza odpowiednie przekształcenie kodu źródłowego, znajdującego się w ogólności w jednym z wielu plików (modułów) współtworzących program, na przemieszczalny kod wynikowy.

Na tym etapie można wyliczyć adresy obiektów i kodu znajdujących się w tym module względem początku modułu. Adresy odnoszące się do obiektów i kodu w innych modułach mogą zostać związane dopiero na etapie konsolidacji.

Systemy operacyjne

Konsolidacja

- Konsolidacja statyczna — odniesienia do innych modułów zamieniane są na adresy w czasie tworzenia programu wykonywalnego.
- Konsolidacja dynamiczna
  - w czasie ładowania — w czasie ładowania programu następuje załadowanie modułów bibliotecznych i związanie odpowiednich adresów,
  - w czasie wykonania — załadowanie modułów bibliotecznych i związanie adresów następuje dopiero przy odwołaniu się do nich w czasie wykonywania programu.

Zarządzanie pamięcią operacyjną (31)

W konsolidacji łączy się wynikowe moduły przemieszczalne, powstałe na etapie kompilacji. Konsolidację statyczną przeprowadza na etapie tworzenia programu ładowalnego. Jeśli jednak te same funkcje wykorzystywane są w wielu programach ładowalnych, każdy plik dla takiego programu będzie zawierał kopie modułów bibliotecznych, obejmujących te funkcje.

W przypadku konsolidacji dynamicznej program ładowalny zawiera tylko informację o tym, jakie moduły biblioteczne potrzebne są do wykonania. Moduły te nie są natomiast kopiowane do pliku z programem, są natomiast dołączane podczas uruchamiania lub wykonania. Można w ten sposób zaoszczędzić miejsce na dysku lub zasoby sieci (pasmo) podczas transmisji takiego programu. Problem może się pojawić, gdy nastąpi przeniesienie programu w inaczej skonfigurowane środowisko, w którym nie wszystkie wymagane moduły biblioteczne są dostępne.

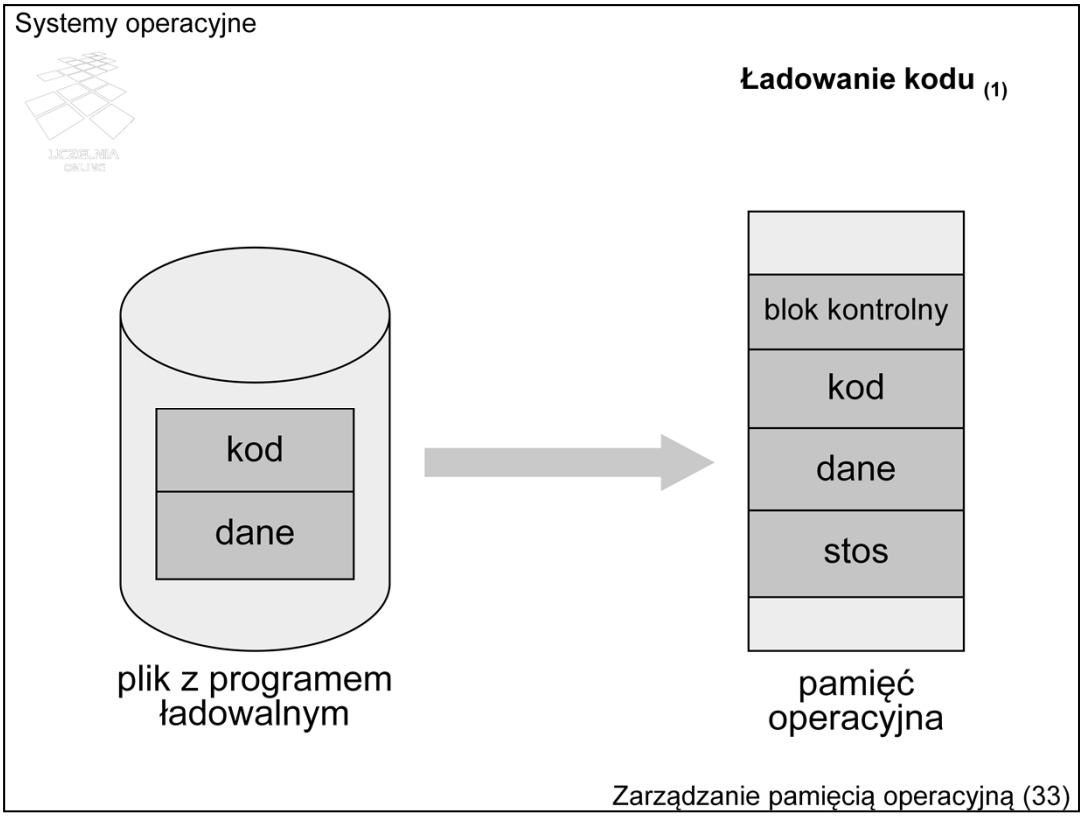
Systemy operacyjne

Konsolidacja statyczna

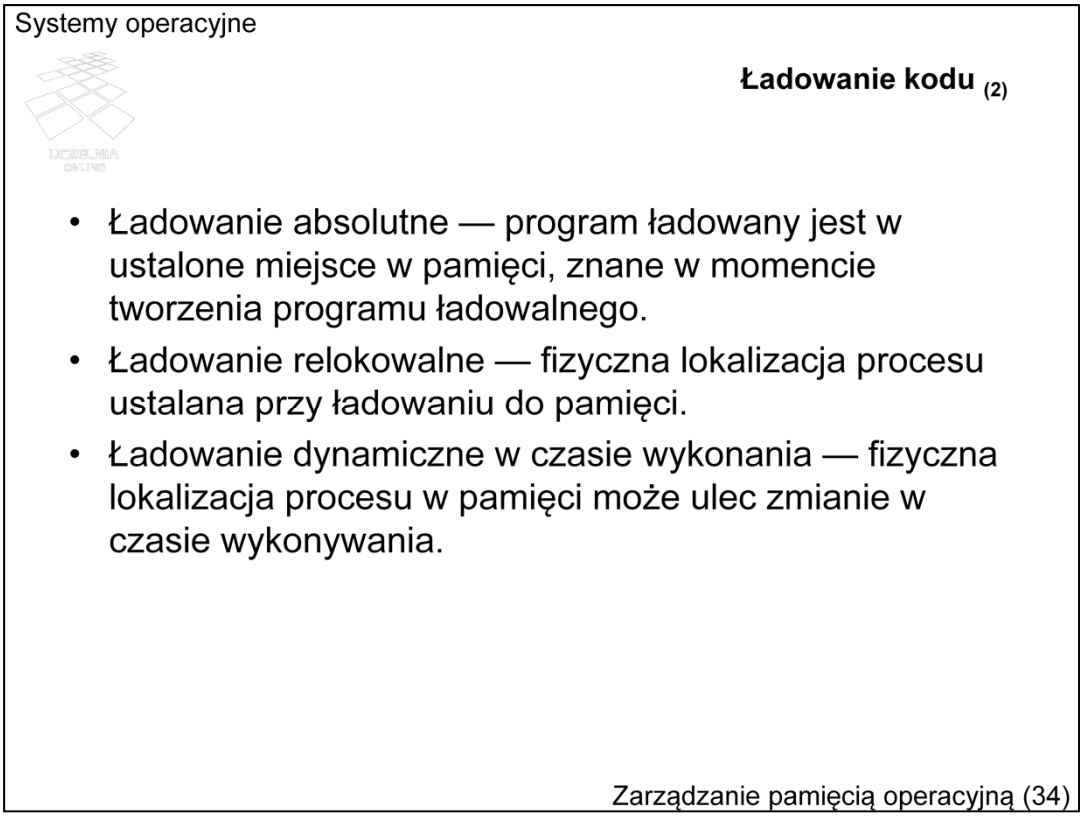
- W czasie łączenia modułów przemieszczalnych w jeden program ładowalny zwany też modułem absolutnym (ang. absolute module), do adresów przemieszczalnych dodawane są wartości, wynikające z przesunięcia danego modułu przemieszczalnego względem początku modułu absolutnego.
- Gdyby docelowa lokalizacja programu w pamięci była z góry znana, na etapie tym mogłyby zostać wygenerowane adresy fizyczne.

Zarządzanie pamięcią operacyjną (32)

Adresy obiektów w modułach przemieszczalnych przeliczane są stosownie do odwzorowania danego moduły w przestrzeni logicznej programu (modułu absolutnego). Do adresów przemieszczalnych dodawane są więc wartości, wynikające z przesunięcia danego modułu przemieszczalnego względem początku modułu absolutnego. Wiązane są również adresy, które odnoszą się do innych modułów wynikowych.



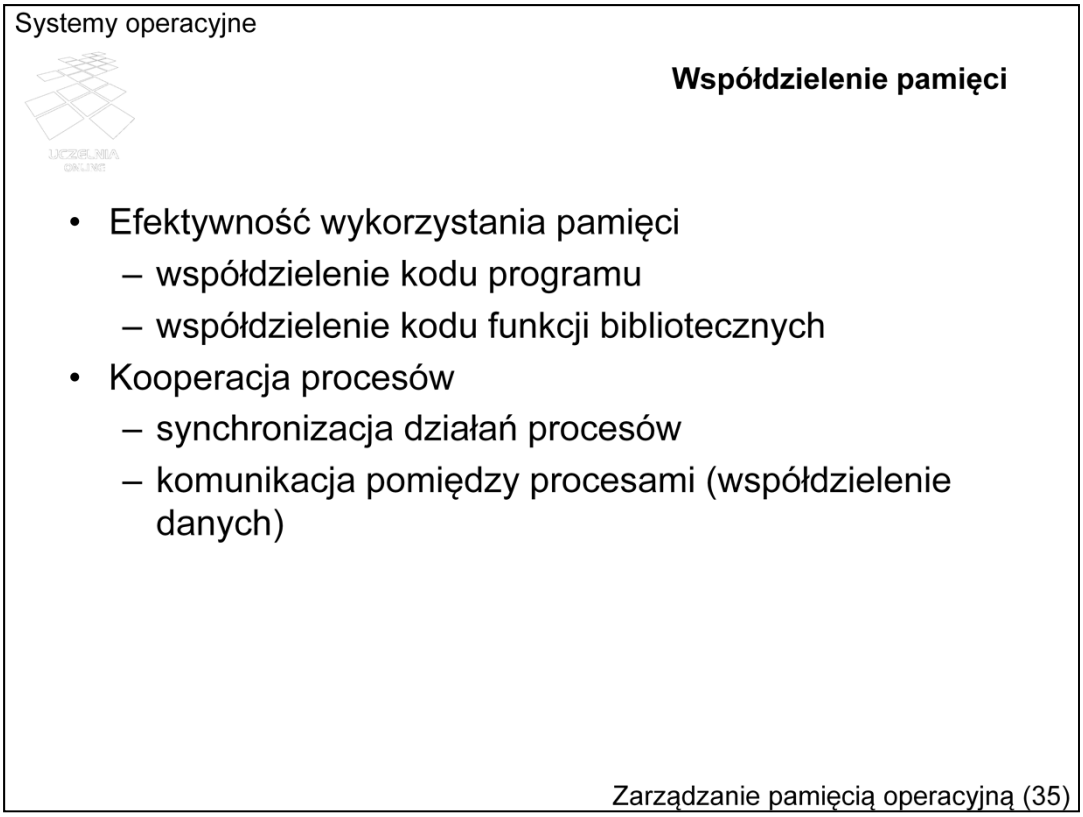
Logiczna przestrzeń adresowa procesu obejmuje kod programu, dane i stos. W przestrzeni tej może być również zlokalizowana część bloku kontrolnego procesu, obejmująca te atrybuty, które potrzebne są tylko w trakcie przetwarzania w kontekście danego procesu (np. u-obszar w systemie UNIX). Część obrazu procesu w pamięci ładowana jest z pliku z programem — kod samego programu oraz dane inicjalizowane w kodzie. Pozostałe części obrazu są tylko opisane (np. rozmiar stosu) i tworzone są przez jądro systemu operacyjnego. Tworzony jest oczywiście również blok kontrolny. Jeśli system dostarcza wsparcia dla współdzielenia, to kod programu lub funkcji bibliotecznych może być współdzielony z innymi procesami.



Ładowanie absolutne oznacza, że lokalizacja procesu w pamięci znana jest zanim nastąpi załadowanie kodu do pamięci. W przypadku takiego ładowania adresy fizyczne mogą być związane na etapie kompilacji lub konsolidacji.

W przypadku ładowania relokowalnego adresy fizyczne ustalane są dopiero na etapie tworzenia obrazu procesu w pamięci, co oznacza, że wszystkie adresy muszą zostać odpowiednio przeliczone. Wymagane jest zatem wskazanie tych miejsc w programie ładowalnym, które zawierają adresy absolutne i dodania do nich przemieszczenia względem początku obszaru pamięci fizycznej. Oprócz czasochłonności samego przeliczania format pliku z programem ładowalnym musi więc dodatkowo uwzględniać identyfikację adresów, przez co staje się skomplikowany i powoduje wzrost rozmiaru samego pliku. Taka forma ładowania komplikuje przemieszczanie kodu w pamięci podczas wykonywania procesu oraz wymianę pomiędzy pamięcią główną a pamięcią pomocniczą, gdyż wymaga przeliczenia adresów przy każdej zmianie lokalizacji procesu w pamięci fizycznej.

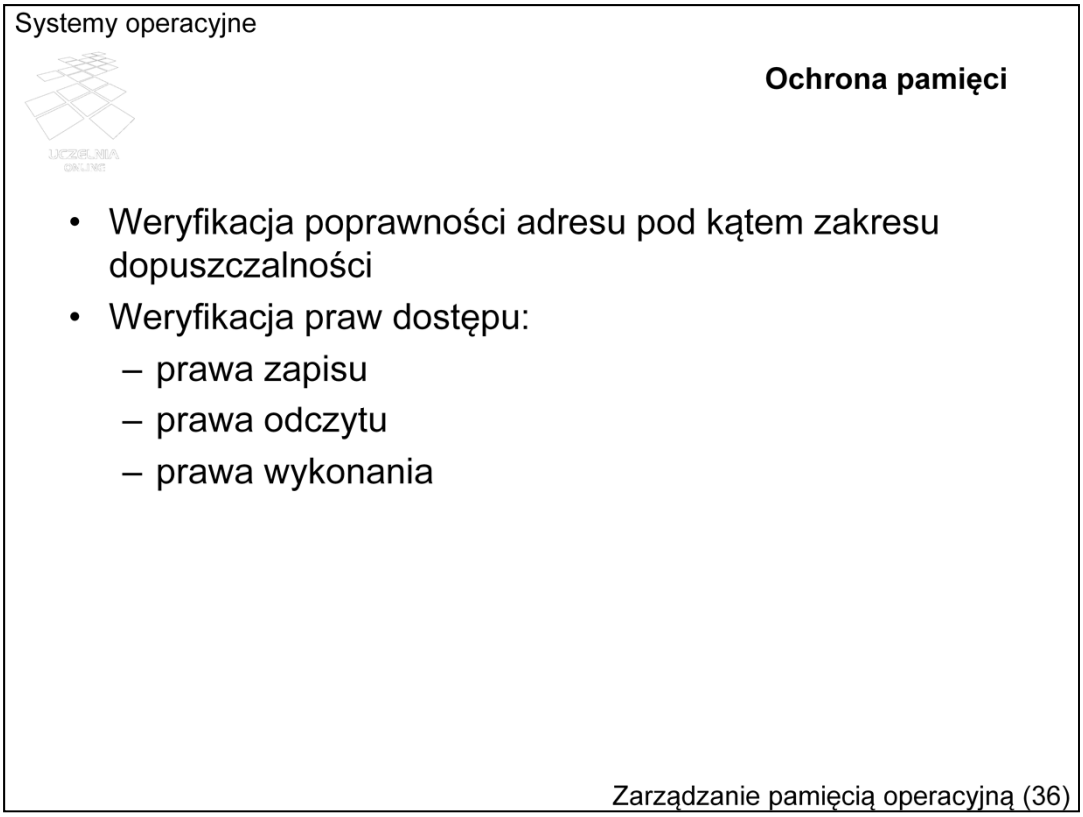
Przy pewnym wsparciu na poziomie architektury komputera można jednak zrealizować ładowanie dynamiczne, w którym adresy fizyczne ustalane są przez jednostkę zarządzania pamięcią dopiero przy odwołaniu do pamięci. Adres wystawiony przez procesor poddawany jest więc odpowiedniej transformacji, zanim zostanie wystawiony na szynie adresowej magistrali systemowej. W takim przypadku można mówić o rozdzieleniu logicznej i fizycznej przestrzeni adresowej.



Jednym z celów współdzielenia pamięci jest poprawa efektywności jej wykorzystania. Dzięki współdzieleniu ten sam obszar pamięci fizycznej, zawierającej program lub dynamicznie ładowane moduły biblioteczne, można odwzorować w obrazy logiczne wielu procesów. Przy braku współdzielenia każdy proces musiałby mieć własną kopię kodu, przechowywaną w osobnym obszarze pamięci.

Pamięć jest najszybszym środkiem przekazywania informacji pomiędzy procesami. Dlatego drugim celem współdzielenia jest dostęp do wspólnego obszaru pamięci, za pośrednictwem którego procesy przekazują sobie sygnały synchronizujące oraz dane do przetwarzania.

Realizacja współdzielenia wymaga rozwiązania podobnych problemów z wiązaniem adresów, jakie pojawiają się przy relokacji, a ponadto uniemożliwia ochronę pamięci lub wprowadza ograniczenia w dostępie. Współdzielenie pamięci przy zachowaniu elastyczności dostępu wymaga rozdzielenia logicznej i fizycznej przestrzeni adresowej oraz możliwości niezależnego odwzorowania poszczególnych części logicznego obrazu procesu na obszar pamięci fizycznej.



Na jednym z pierwszych slajdów pojawiła się informacja na temat ochrony pamięci, ograniczonej do weryfikacji poprawności zakresu dla adresu wystawionego przez procesor. Obraz procesu jest jednak skomplikowany — w jego skład wchodzą między innymi kod programu, dane oraz stos. Stos może być zarówno zapisywany, jak i odczytywany. Dane, zależnie od rodzaju, mogą być albo tylko czytane, albo czytane i zapisywane. Kod programu na ogół nie jest modyfikowany, a jest to niedopuszczalne w przypadku jego współdzielenia. W związku z tym można wprowadzić dodatkowe restrykcje odnośnie dostępu do pamięci. Poszczególne części obrazu procesu muszą mieć pewne atrybuty, wskazujące na rodzaj zawartości i wynikające stąd prawa dostępu. Jednostka zarządzania pamięcią, weryfikująca poprawność dostępu, musi mieć z kolei informacje o realizowanym właśnie cyklu maszynowym. Dostęp do zawartości pamięci z kodem możliwy jest tylko w fazie (cyklu maszynowym) pobrania kodu rozkazu. W fazie tej nie jest z kolei możliwy dostęp do obszarów, które nie są oznaczone jako kod. W ten sposób niemożliwa jest również modyfikacja kodu programu. Obszar danych tylko do odczytu nie jest z kolei dostępny w cyklu maszynowym zapisu pamięci. Ogólnie obszar danych i stosu nie jest dostępny w fazie pobrania kodu rozkazu, co wynika z wcześniejszych wyjaśnień. Restrykcje takie wprowadzana są często na potrzeby bezpieczeństwa. Np. przejęcie sterowania w procesie poprzez tzw. przepełnienie bufora, możliwe jest dzięki temu, że zawartość obszaru danych lub stosu interpretowana jest jako ciąg instrukcji. Ochrona na tym poziomie wymaga jednak wyodrębnienia odpowiednich części programu i właściwego ich opisania, co wymaga wsparcia sprzętowego.

Systemy operacyjne

Stronicowanie

- Arbitralny podział pamięci fizycznej na ramki, w które ładowane są odpowiednie strony obrazu procesu.
- Podział logicznej przestrzeni adresowej na strony o takim samym rozmiarze, jak ramki w pamięci fizycznej.
- Zalety:
  - brak problemu fragmentacji zewnętrznej,
  - wspomaganie dla współdzielenia i ochrony pamięci.
- Wady:
  - narzut czasowy przy transformacji adresu,
  - narzut pamięciowy (na potrzeby tablicy stron),
  - fragmentacja wewnętrzna (niewielka).

Zarządzanie pamięcią operacyjną (37)

W dotychczas rozważanych aspektach istniało domniemanie, że obraz procesu zajmuje ciągły obszar pamięci fizycznej. Ewentualne odwzorowanie obrazu logicznego na fizyczny polegało na dodaniu do adresu logicznego przemieszczenia, wynikającego z przesunięcia początku obszaru pamięci procesu względem początku pamięci fizycznej.

Obraz procesu można jednak podzielić na odrębne części i dla każdej części zdefiniować odwzorowanie. Jednym z tego typu podejść jest stronicowanie (ang. paging), w którym obraz procesu oraz pamięć fizyczna dzielone są na równe obszary o ustalonej wielkości zwane stronami (ang. pages). W celu odróżnienia stron z obrazem procesu od stron pamięci fizycznej te ostatnie nazywa się *ramkami* (ang. frames).

Strony i tym samym ramki mają współcześnie rozmiar od kilku do kilkudziesięciu kilobajtów, są więc stosunkowo niewielkie w stosunku do rozmiary obrazu procesu, czy dostępnej pamięci fizycznej. Stronicowanie jest więc pewną formą podziału stałego, jednak obraz procesu może zająć kilka jednostek, wynikających z tego podziału. Procesowi można przydzielić ramki rozmieszczone w dowolnym miejscu dostępnego obszaru pamięci fizycznej (nie muszą to być kolejne, sąsiadujące ze sobą jednostki). W ten sposób kosztem pewniej fragmentacji wewnętrznej rozwiązuje się problem fragmentacji zewnętrznej.

Systemy operacyjne

Stronicowanie — transformacja adresu

- Adres logiczny zawiera numer strony i przesunięcia na stronie (ang. offset), np.:

nr strony (22 bity)	przesunięcie (10 bitów)

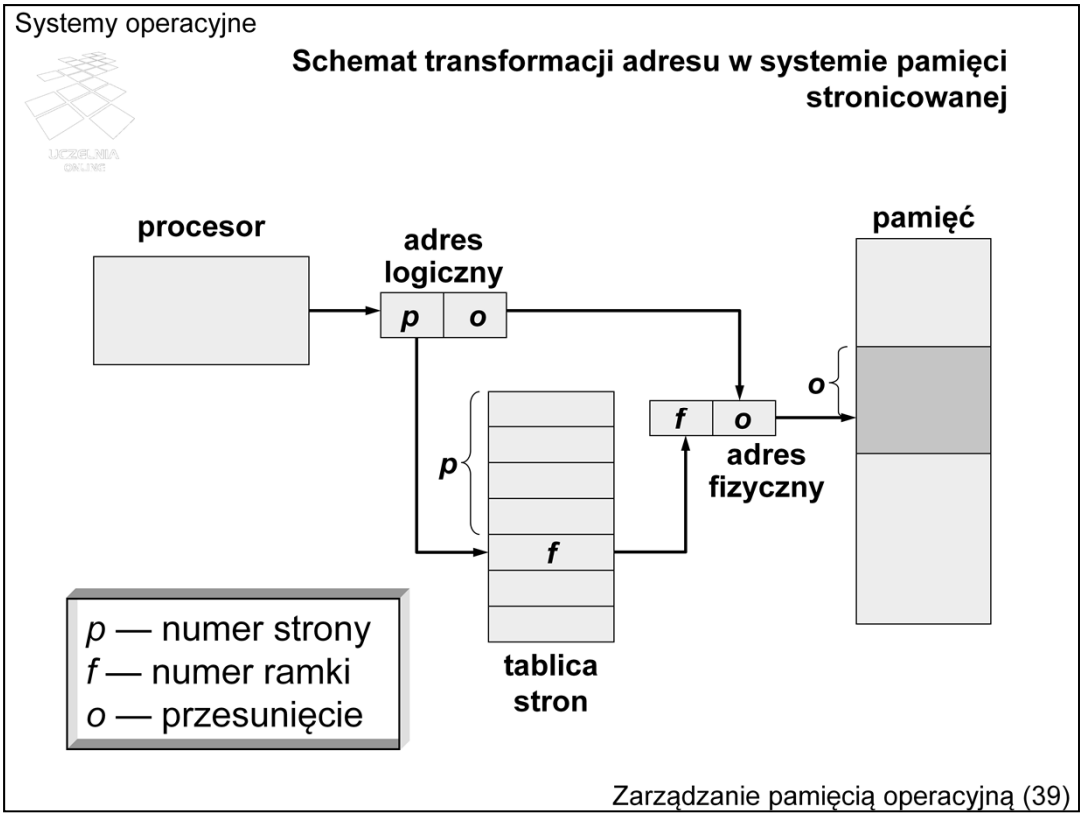
- Transformacja adresu polega na zastąpieniu numeru strony numerem ramki.
- Odwzorowanie numeru strony na numer ramki wykonywane jest za pomocą tablicy stron (ang. page table).

Zarządzanie pamięcią operacyjną (38)

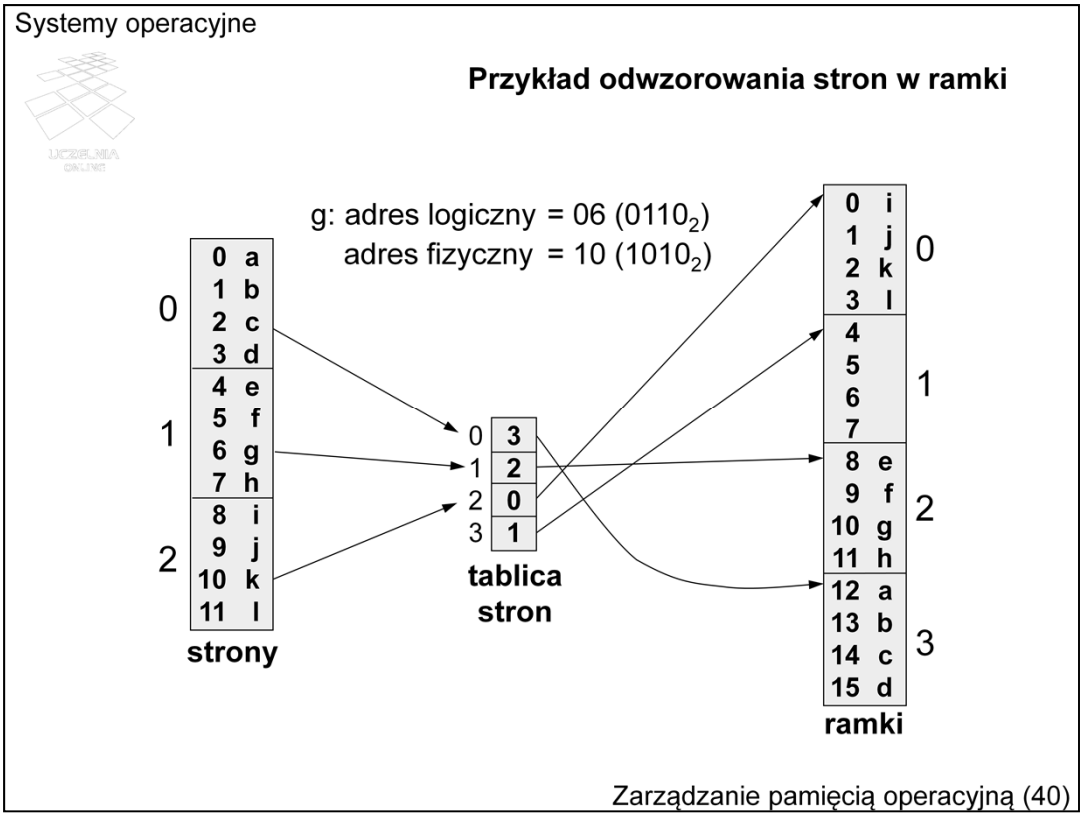
Mniej znaczące bity w adresie logicznym traktowane są jako przesunięcie wewnątrz strony (tym samym ramki), podczas gdy pozostałe (bardziej znaczące) bity wyznaczają numer strony. W przedstawionym przykładzie adresu logicznego na przesunięcie przeznaczono 10 bitów, co oznacza, że rozmiar strony/ramki wynosi 2^{10} = 1 KB.

Każda strona ma ustalony numer ramki. Informacja o numerach ramek dla poszczególnych stron znajduje się w tablicy stron. Tablica stron zlokalizowana jest w pamięci fizycznej i musi być dostępna dla jednostki zarządzania pamięcią, której zadanie w ramach transformacji adresu sprowadza się do zastąpienia numeru strony w adresie logicznym numerem ramki.

W systemach wielozadaniowych każdy proces ma własną tablicę stron, której adres zlokalizowany jest w odpowiednim rejestrze jednostki zarządzania pamięci i podlega zmianie przy przełączaniu kontekstu.

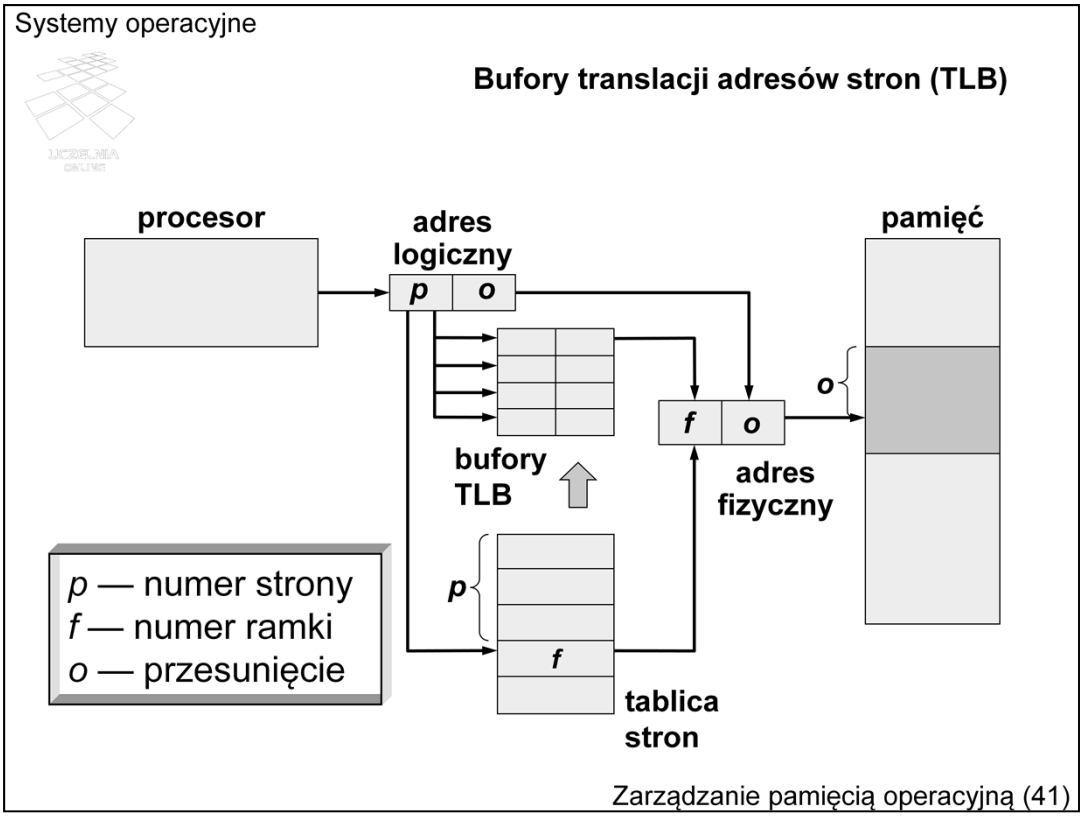


Po wystawieniu adresu logicznego przez procesor jednostka zarządzania pamięcią identyfikuje numer strony i traktując go jako indeks w tablicy stron, lokalizuje odpowiedni wpis. Następnie zamienia numer strony w adresie logicznym na odczytany numer ramki, przesunięcie pozostawiając bez zmian i wystawia taki adres na magistrali systemowej.

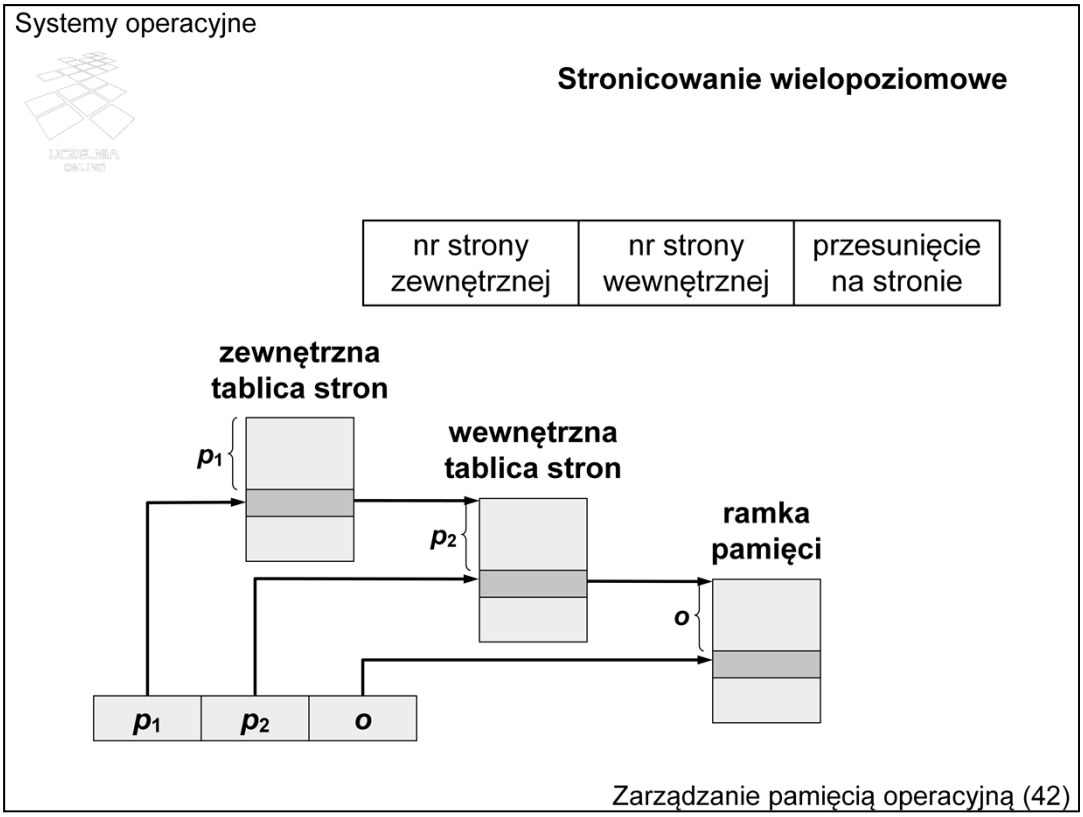


Przykład pokazuje sens stronicowania. W obrazie logicznym (po lewej) pod adresami do 0 do 11 znajdują się kody kolejnych znaków (liter alfabetu). W obrazie fizycznym kolejność jest zupełnie inna, a dane nie zajmują nawet ciągłego obszaru. Przykład adresowania litery *g* obrazuje transformację. Adres logiczny 6 (dziesiętnie) po zastąpieniu dwóch bardziej znaczących bitów (wartości 1) zgodnie z zawartością tablicy stron na pozycji 1 zamieniany jest na 10 (dziesiętnie).

Przykład ma charakter poglądowy, dlatego rozmiar strony wynosi zaledwie 4 bajty. W żadnym rzeczywistym rozwiązaniu nie jest to tak mała jednostka.



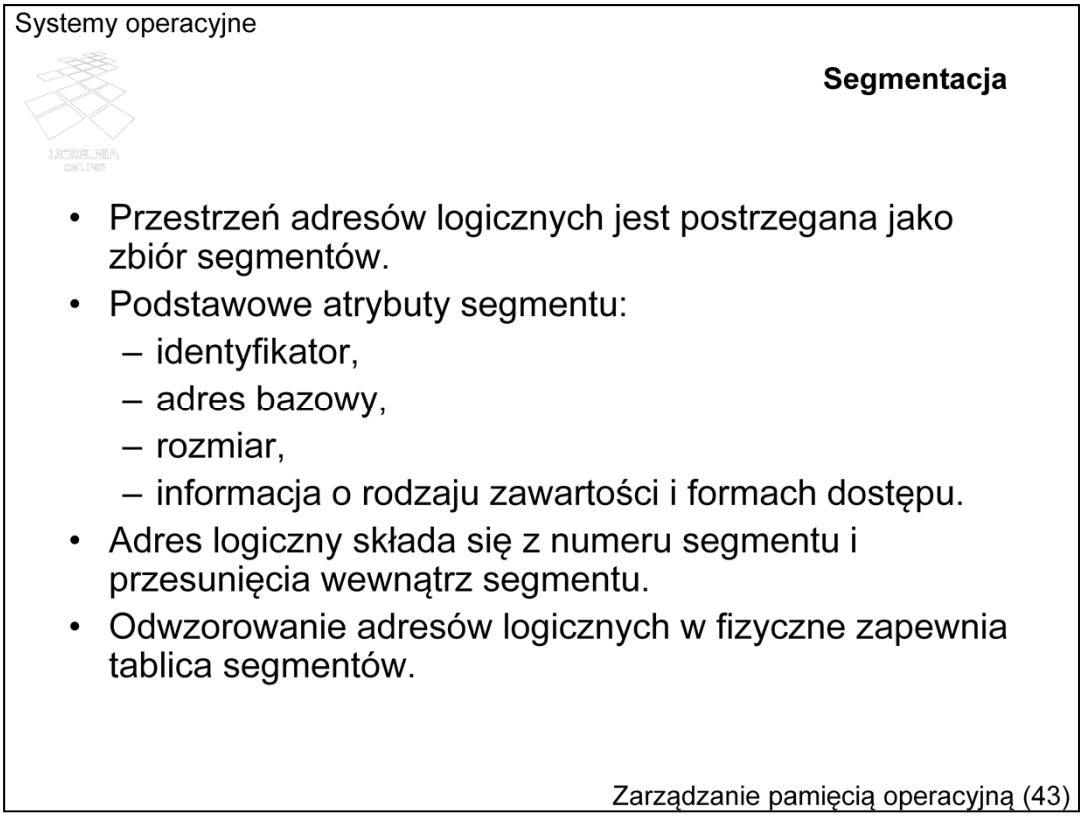
Transformacja adresu wymaga dodatkowego dostępu do pamięci w celu pobrania z tablicy stron informacji o adresowanej stronie. Czas dostępu do właściwej zawartości (kod rozkazu, operandów) wydłuża się więc dwukrotnie. W celu zredukowania dodatkowego obciążenia czasowego stosowana jest szybka pamięć asocjacyjna, zwana buforami TLB (ang. translation look-aside buffer), w której przechowywane są ostatnio pobrane wpisy z tablicy stron. Kluczem, na podstawie którego lokalizowana jest pozycja w buforze TLB, jest numer strony, a wartością na wyjściu jest numer ramki. Jeśli wpis o danym kluczu nie zostanie znaleziony w buforze TLB, następuje normalne odwołanie do tablicy stron, przy czym wpis przenoszony jest do bufor TLB. Wykorzystywana jest tutaj właściwość programów, zwana *lokalnością odniesień do pamięci*.



Przedstawiony wcześniej przykładowy podział 32-bitowego adresu logicznego na 10-bitowe przesunięcie oraz 22 bitowy numer strony oznaczałby, że potencjalnie może być potrzebnym 2^{22} wpisów w tablicy stron. Zakładając, że każdy wpis wymaga 32 bitów (4 bajtów), na tablicę stron potrzebny byłby ciągły obszar pamięci fizycznej o rozmiarze 2^{24} = 16 MB. Znalezienie tak dużego ciągłego obszaru może być kłopotliwe, a rozwiązaniem problemu może być zastosowanie podejścia wielopoziomowego, zwanego również hierarchicznym. W podejściu dwupoziomowym w adresie logicznym wyodrębnia się 3 zakresy bitów:

- numer pozycji w zewnętrznej tablicy stron, opisującej ramkę z wewnętrzną tablicą stron,
- numer pozycji w wewnętrznej tablicy stron, opisujący ramkę z fragmentem (stroną) obrazu procesu,
- przesunięcie wewnątrz ramki z fragmentem obrazu procesu.

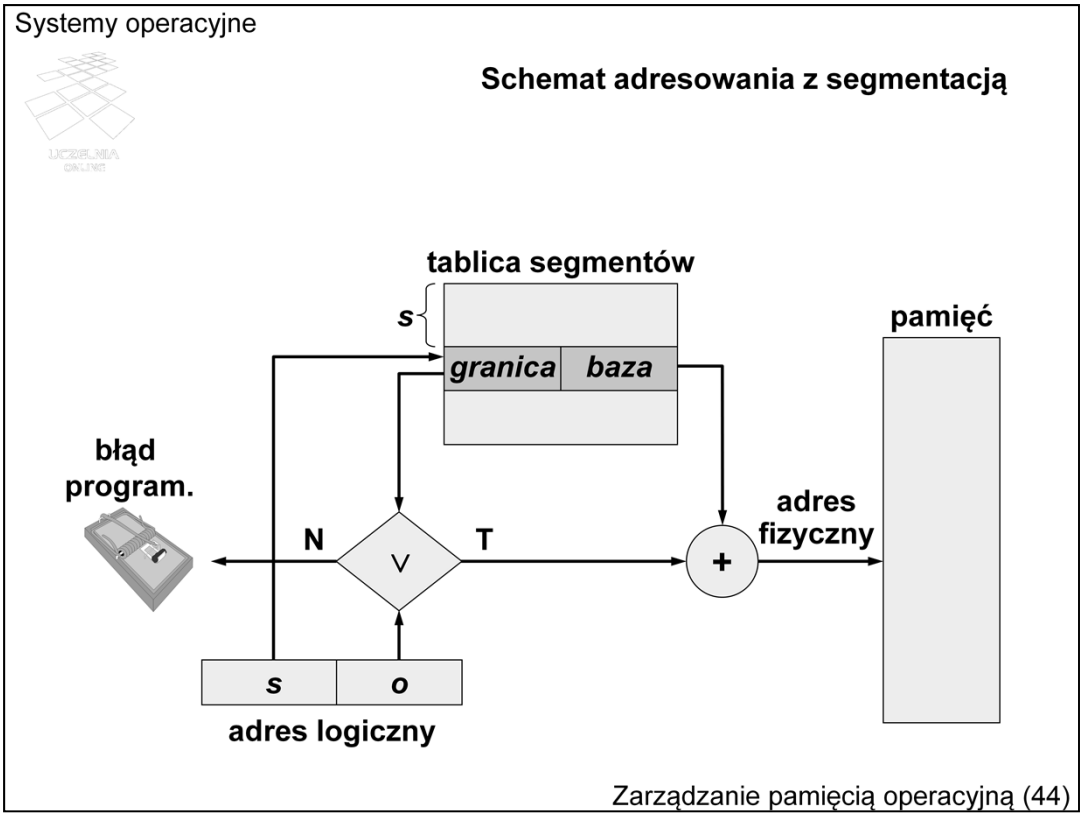
Podejście takie zastosowano między innymi w architekturze Intel IA-32. Zewnętrzną tablicę stron określa się jako *katalog stron*, a wewnętrzną po prostu jako tablicę stron. Na identyfikację pozycji w katalogu stron oraz w tablicy stron przeznaczone jest po 10 bitów z 32-bitowego adresu. Na przesunięcie na stronie pozostaje zatem 12 bitów, co oznacza, że rozmiar strony wynosi 4 KB. W architekturze Sparc zastosowano nawet podeście trzypoziomowe.



Podstawą stronicowania jest stały podział pamięci. Segmentacja (ang. segmentation) oparta jest z kolei na podziale dynamicznym. Obraz procesu dzielony jest na logiczne części, odpowiadające poszczególnym sekcjom programu — sekcji kodu, danych, stosu itp. Dla każdej sekcji definiowany jest odpowiedni segment. Na potrzeby segmentu przydzielane są dynamiczne partycje w pamięci. Segmenty danego procesu mogą być dowolnie rozmieszczone w pamięci, ale każdy segment zajmuje ciągły obszar pamięci.

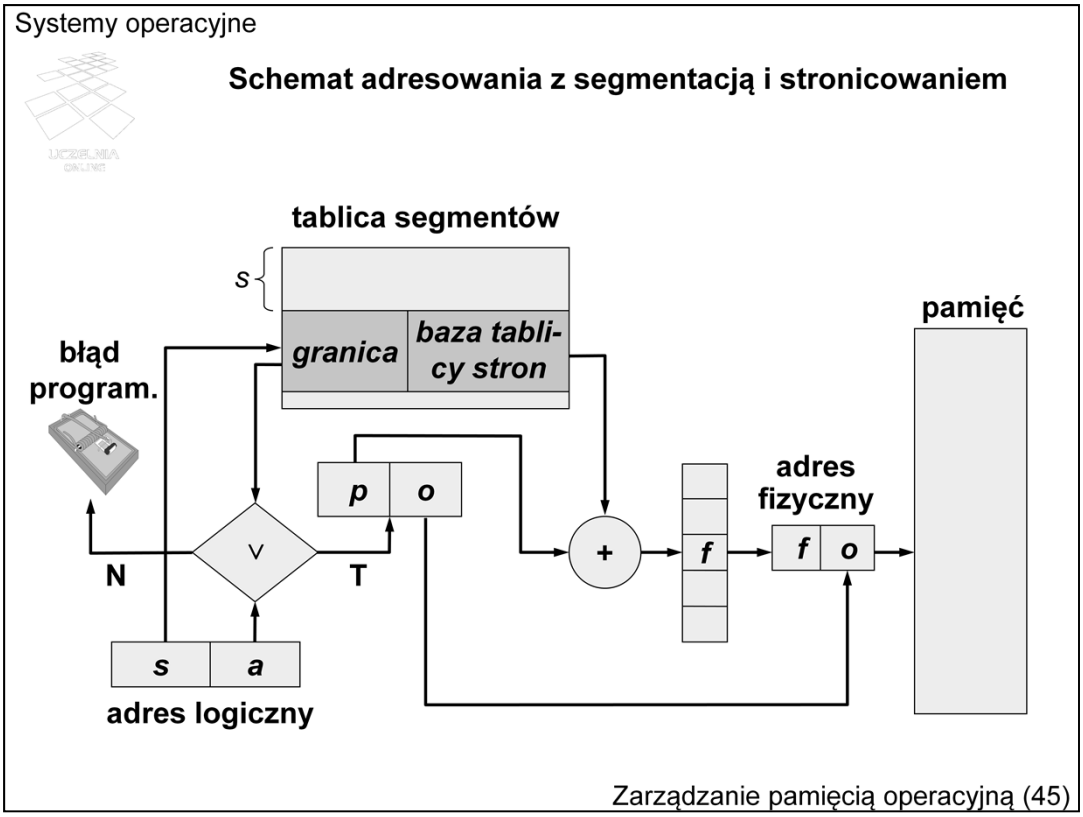
Segment, przeznaczony na określoną sekcję programu, może mieć zróżnicowany rozmiar i zawartość, a także dowolne położenie w pamięci fizycznej. Dlatego opis segmentu (deskryptor) obejmuje takie dane, jak:

- adres bazowy — fizyczny adres początku segmentu w pamięci,
- rozmiar — długość segmentu w ustalonych jednostkach (np. w bajtach, paragrafach),
- atrybuty określające rodzaj zawartości i dostępność (np. kod programu, dane tylko do odczytu, stos itp., pierścień ochrony) — na potrzeby weryfikacji poprawności odniesień,
- identyfikator (określany też jako nazwa lub numer) — wartość wskazująca na opis segmentu w tablicy segmentów (najczęściej indeks w tablicy segmentów). Jeśli identyfikatorem segmentu jest indeks, to jego wartość nie jest przechowywana w deskryptorze, ale wynika z lokalizacji deskryptora w tablicy.



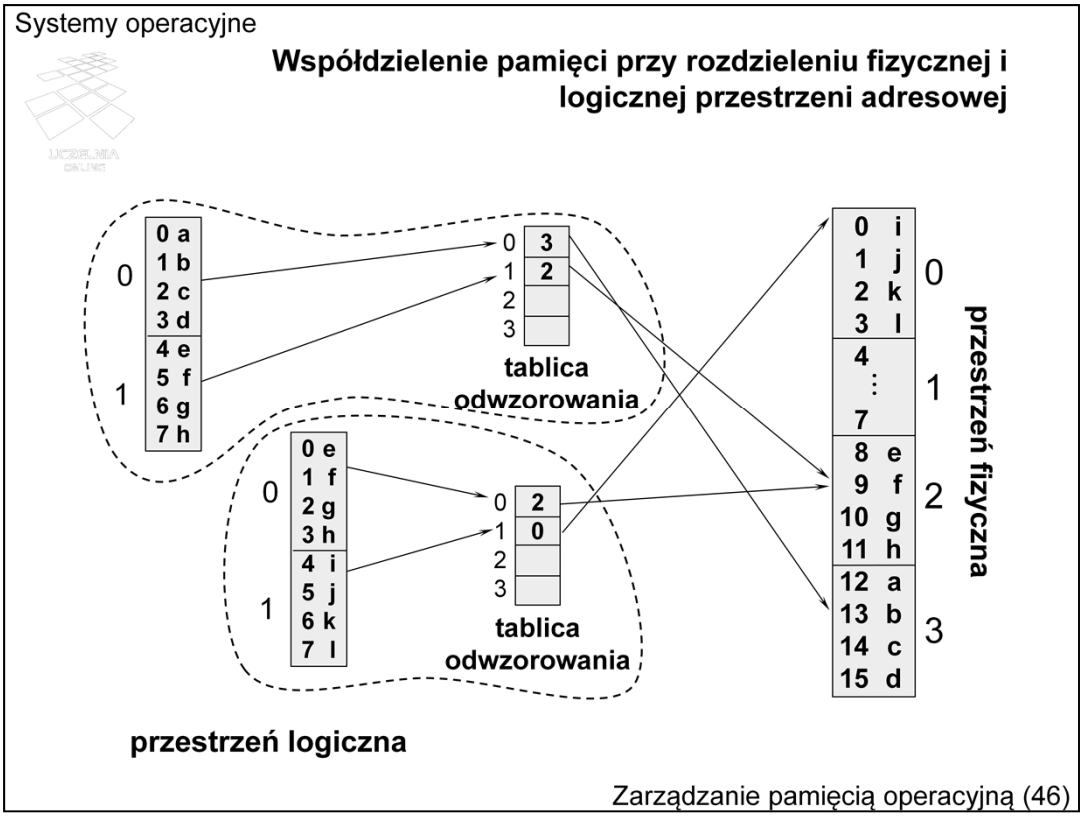
Strona miała rozmiar dokładnie dostosowany do przesunięcia, nie było więc ryzyka operowania przesunięciem naruszającym rozmiar strony. Dla segmentu rozmiar jest jednym z atrybutów niezależnych od przesunięcia. Możliwy jest zatem przypadek użycie przesunięcia, wykraczającego poza rozmiar segmentu. Taki przypadek musi być oczywiście wykryty i zasygnalizowany przez odpowiednie przerwanie diagnostyczne. Stosownie do atrybutów segmentu weryfikowany może być również cykl maszynowy oraz poziom (pierścień) ochrony.

Po poprawnej weryfikacji przesunięcie dodawane jest do adresu bazowego segmentu i powstaje adres fizyczny, wystawiany na magistrali systemowej.



W przeciwieństwie do stron, segmenty mogą mieć dość duży rozmiar, a więc znalezienie dla nich ciągłego obszaru pamięci fizycznej może być kłopotliwe. Chociaż możliwa jest relokacja segmentu, operacja może być czasochłonna. Zalety segmentacji i stronicowania można jednak połączyć, stosując obie techniki. Pamięć w takim podejściu postrzegana jest jako zbiór segmentów, które składają się z ramek. Przesunięcie wewnątrz segmentu traktowana jest jako adres w pamięci stronicowanej, czyli adres składający się z numeru strony oraz przesunięcia wewnątrz strony. Zamiast adresu bazowego segmentu w deskryptorze znajduje się adres tablicy stron danego segmentu, za pośrednictwem której ustalany jest numer ramki.

Nie jest to jedyny sposób transformacji adresu. W architekturze IA-32 (począwszy już od procesora 80386) zastosowano podejście, w którym każdy adres poddawany jest transformacji zgodnie z zasadą segmentacji, w wyniku czego powstaje adres liniowy. Jeśli procesor pracuje w trybie z pamięcią stronicowaną, adres liniowy poddawany jest dalszej transformacji, właściwej dla stronicowania dwupoziomowego.



Współdzielenie pamięci można zrealizować w sensowny sposób dopiero wówczas, gdy określony fragment logicznej przestrzeni adresowej można odwzorować na współdzielony fragment pamięci niezależnie od pozostałej części przestrzeni logicznej. Zarówno stronicowanie, jak i segmentacja udostępniają środki do takiego odwzorowania, gdyż wpisy w odpowiednich tablicach (stron lub segmentów) poszczególnych procesów mogą się odnosić do tego samego obszaru pamięci fizycznej. Weryfikacja poprawności odwołania, związana z ochroną, może być przeprowadzona na poziomie adresów logicznych.