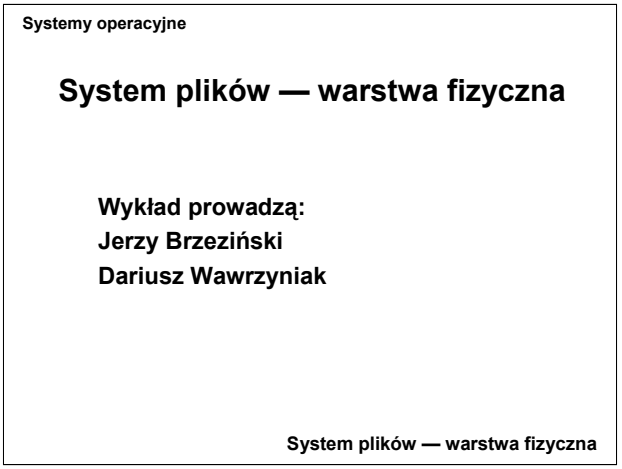
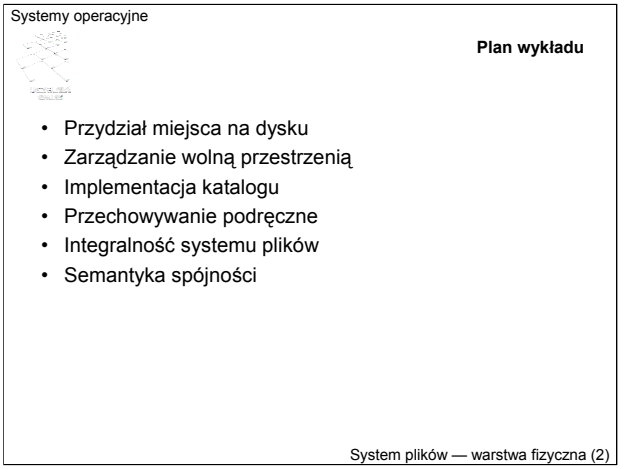
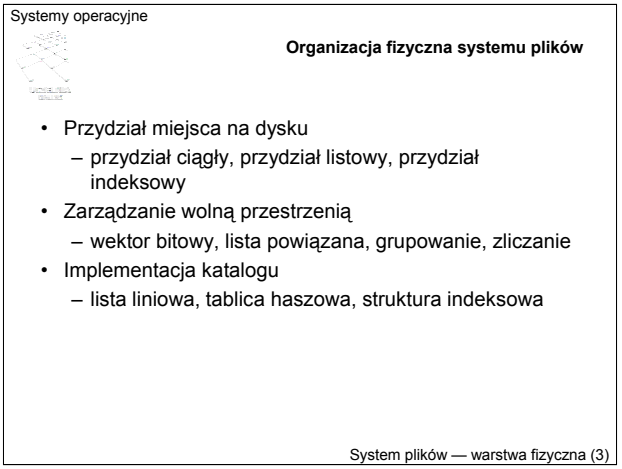


Celem wykładu jest prezentacja różnych podejść do implementacji systemu plików. Podejścia opierają się na założeniu, że urządzeniem składowania danych jest dysk i dotyczą one przede wszystkim organizacji przestrzeni dyskowej. Podkreślane są wady i zalety tych podejść w kontekście różnych form dostępu do danych w plikach.



Zasadniczą kwestią, przedstawianą w ramach wykładu, jest organizacja przestrzeni dyskowej, obejmująca powiązanie pliku z blokami dyskowymi oraz zarządzanie blokami wolnymi. Poruszona jest też implementacja katalogu. W kolejnej części omawiany jest sposób realizacji operacji plikowych oraz wynikający z tego problem integralności danych. Na koniec zasygnalizowana jest problematyka współbieżnego dostępu do pliku.

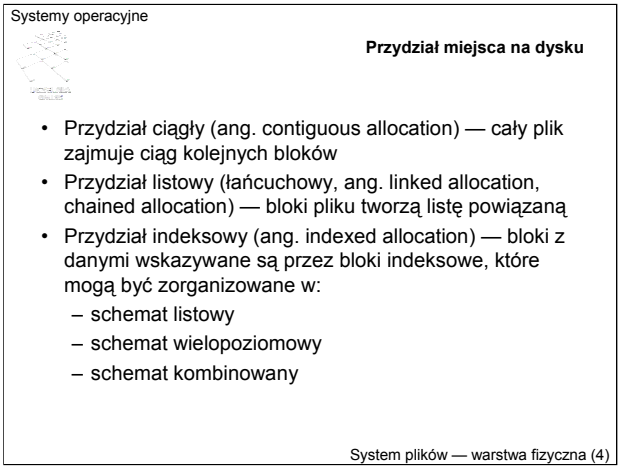


Przestrzeń dyskowa na potrzeby systemu plików zorganizowana jest w jednostki alokacji, zwane krótko blokami. Blok jest wielokrotnością sektora dysku.

W zakresie przydziału miejsca dla pliku na dysku, czyli powiązania bloków z plikiem, omówione są trzy podejścia: przydział ciągły, listowy i indeksowy, przy czym ten ostatni wymaga dodatkowo określenia sposobu powiązania bloków indeksowych. Podejściem opartym na idei przydziału listowego jest tablica alokacji plików (FAT).

Wolna przestrzeń jest dopełnieniem przestrzeni przydzielonej dla plików w ramach danej strefy dysku. Jednak efektywna identyfikacja wolnych bloków na potrzeby tworzenia nowych plików lub rozszerzania istniejących wymaga utrzymywania komplementarnej informacji. Wyróżnia się cztery rodzaje takich struktur: wektor bitowy, lista powiązana, grupowanie, zliczanie, przy czym idee niektórych podejść można łączyć.

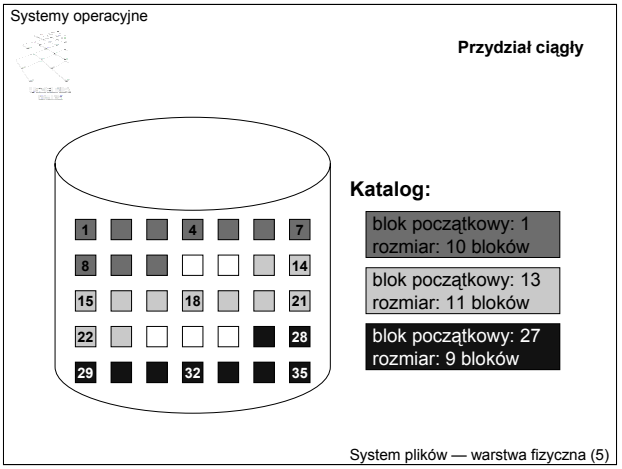
Dla katalogu przydzielane są bloki tak, jak dla innych plików dyskowych. Jest to zatem również plik, ale jego struktura jest odpowiednio definiowana przez system. Specyficzne są również operacje dostępu. W przypadku niewielkiej liczby wpisów katalog może być zwykłą listą (czy tablicą). Czas wyszukiwania zależy wówczas liniowo do liczby wpisów, co staje się nie akceptowalne w przypadku rozrastających się katalogów. Dla dużych katalogów lepiej jest zastosować jakąś strukturę przyspieszającą wyszukiwanie informacji, np. B/B^+-drzewo lub tablicę haszową.



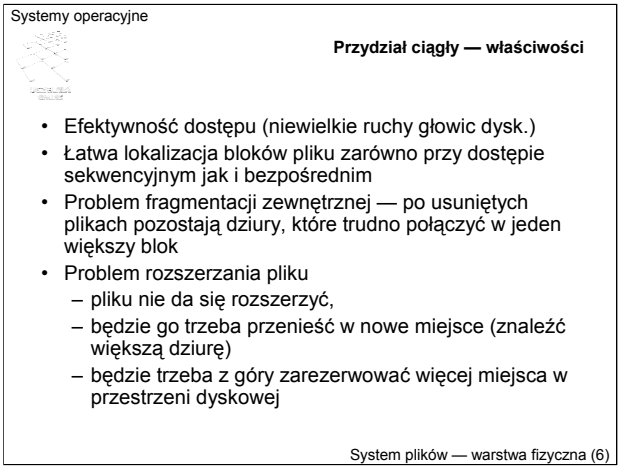
Przydział ciągły oznacza, że zawartość pliku umieszczona jest w kolejnych według jakieś jednoznacznej numeracji blokach (jednostkach alokacji). Numeracja związana jest z fizycznym położeniem bloku na urządzeniu.

W przydziale listowym poszczególne bloki powiązane są wskaźnikami. W przypadku listy jednokierunkowej w każdym bloku przechowywany jest wskaźnik (czyli numer, indeks) bloku następnego. Dzięki temu zawartości pliku może być rozmieszczona w blokach, dowolnie rozproszonych w dostępnej przestrzeni dyskowej.

W przydziale indeksowym numery bloków dyskowych z danymi skupione są w jednym miejscu, w tzw. bloku indeksowym. Jeśli jeden blok indeksowy jest zbyt mały, żeby pomieścić numery wszystkich bloków z danymi, tworzone są kolejne bloki indeksowe i organizowane w schemat listowy, wielopoziomowy lub kombinowany.



W przedstawionym przykładzie przydziału ciągłego atrybut *lokalizacja* pliku implementowany jest za pomocą dwóch wartości: numeru (indeksu) pierwszego bloku oraz liczby bloków, która wynika z rozmiaru pliku. Pierwszy plik (zielony) zajmuje więc 10 bloków począwszy od bloku 1, drugi plik (czerwony) zajmuje 11 bloków począwszy od bloku nr 13 (czyli do 23 włącznie), a trzeci plik (żółty) zajmuje 9 bloków od bloku nr 27.

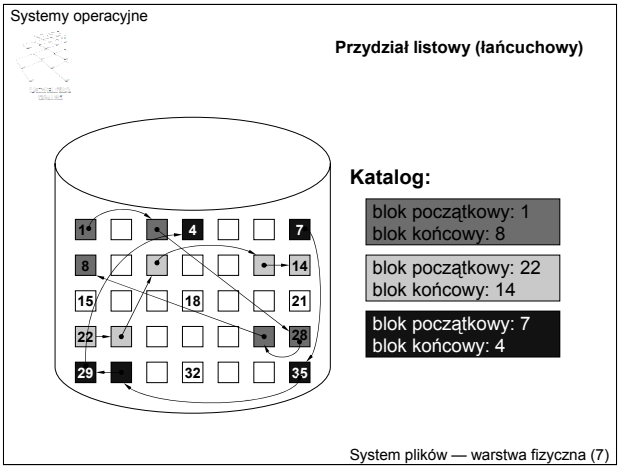


Niewątpliwą zaletą przydziału jest efektywność. Dotyczy ona zarówno złożoności czasowej operacji dostępu, jak i złożoności struktur danych. W przypadku wielu operacji dostępu do tego samego pliku można oczekiwać niewielkich ruchów głowic dyskowych, a więc krótkiego czasu wyszukiwania. Łatwo można zlokalizować dowolny fragment pliku. Identyfikacja bloków pliku nie wymaga żadnych dodatkowych danych poza dwoma wartościami całkowitymi (numerem pierwszego bloku i rozmiarem).

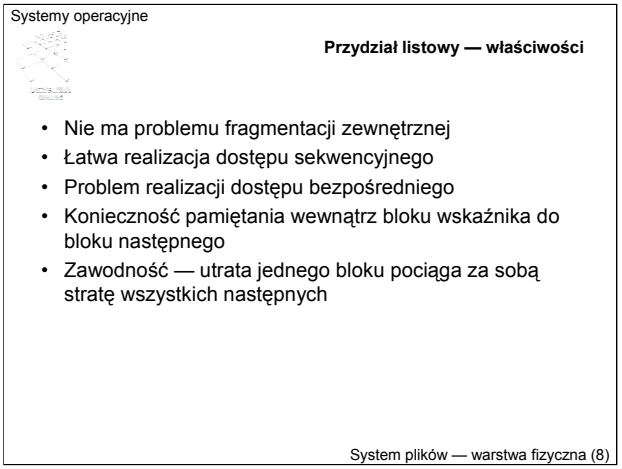
Wadą rozwiązania jest trudność obsługi fragmentacji zewnętrznej. W przedstawionym przykładzie pomiędzy pierwszym i drugim plikiem (zielonym i czerwonym) są dwa bloki wolne, a pomiędzy drugim i trzecim plikiem (czerwonym i żółtym) są trzy bloki wolne. Potencjalnie można by jeszcze znaleźć miejsce na plik o rozmiarze pięciu bloków, ale **nie stanowią one ciągłego obszaru**. Przenoszenie bloków natomiast jest operacją w ogólności czasochłonną. Można ją wykonać od czasu do czasu w ramach optymalizacji przestrzeni dyskowej, ale nie sposób robić tego przy każdym problemie alokacji bloków.

Nie ma też dobrej ogólnej metody na rozszerzanie pliku.

Wymienione wady dyskwalifikują to podejście w systemach plików, w których często wykonywane są modyfikacje zawartości i towarzyszące temu skracanie lub rozszerzanie pliku. Podejście można natomiast stosować w systemach, gdzie modyfikacje są sporadyczne.



W przedstawionym przykładzie atrybut lokalizacja w katalogu (lub w związanej z nim strukturze) składa się z dwóch indeksów — indeksu pierwszego i ostatniego bloku (alternatywnie rozmiar pliku). Jednak pozostałe indeksy ukryte są blokach z danymi. W każdym bloku musi być zarezerwowane miejsce na indeks następnego bloku. Pierwszy plik (zielony) rozpoczyna się od bloku 1. W bloku pierwszym oprócz danych (treści pliku) przechowywany jest indeks bloku następnego, czyli 3. W bloku 3 z kolei przechowywany jest indeks bloku następnego, czyli 28 itd.

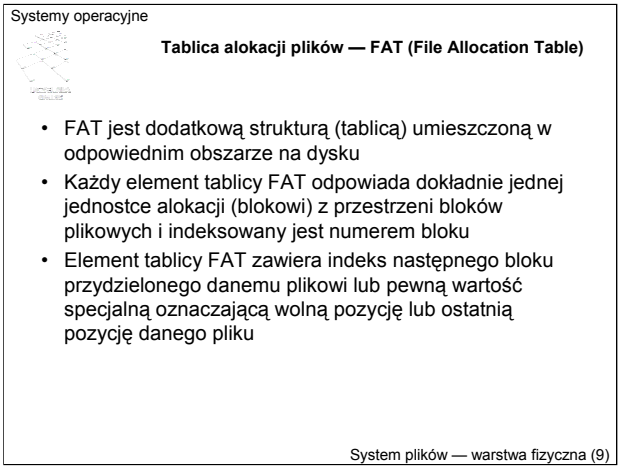


Przydział listowy rozwiązuje problem fragmentacji zewnętrznej i wynikających z niej ograniczeń na przydział bloków.

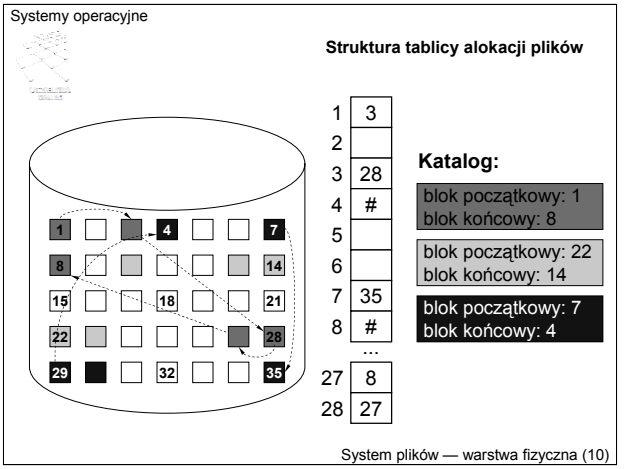
Łatwo realizowalny jest dostęp sekwencyjny, jeśli przebiega się plik od początku do końca. Problem może powstać, gdy trzeba się cofnąć. Jeśli dane są w poprzednim bloku, należy rozpocząć wyczytywanie bloków od początku. W przypadku próby dostępu bezpośredniego niejednokrotnie również należy rozpocząć przeglądanie bloków od początku, dlatego ten rodzaj dostępu jest problematyczny w realizacji.

Kosztem realizacji tego typu podejścia jest konieczność przeznaczenia pewnej ilości miejsca w bloku na przechowywanie wskaźnika do bloku następnego.

W przypadku uszkodzenia sektora dysku jest ryzyko utraty bloku. W przypadku przydziału listowego może to mieć większe konsekwencje, gdyż utrata bloku oznacza również utratę indeksu bloku następnego i tym samym wszystkich pozostałych bloków aż do końca pliku.

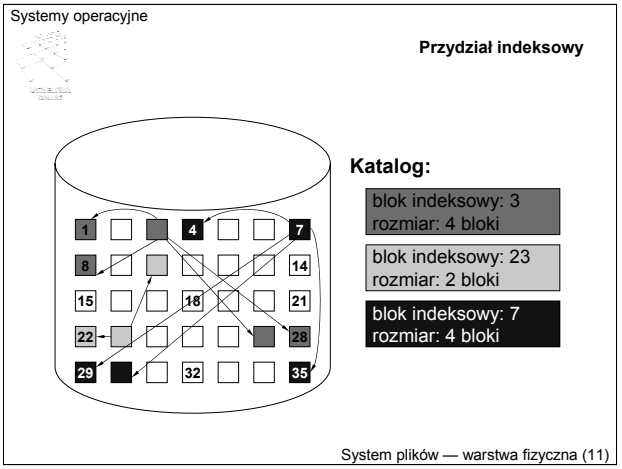


FAT jest odmianą przydziału listowego, w której sama lista powiązań jest oddzielna od bloków z danymi. Indeks następnego bloku nie jest przechowywany w bloku z danymi, tylko na odpowiadającej temu blokowi pozycji w tablicy FAT. Dzięki temu w jednej operacji dostępu do dysku można wczytać do pamięci większy fragment łańcucha powiązań i sprawniej zrealizować dostęp bezpośredni.

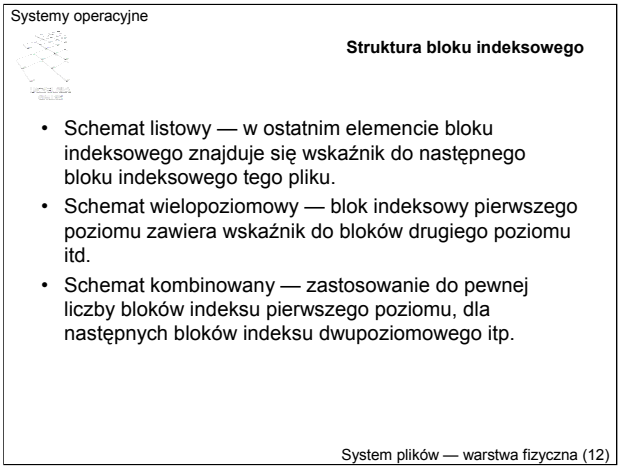


Zaprezentowany przykład jest adaptacją przykładu z przydziałem listowym. Tablica FAT na pozycji nr 1 zawiera wartość 3, co oznacza, że następnym blokiem po bloku nr 1 jest blok nr 3. Na pozycji 3 w tablicy FAT jest wartość 28, a więc blok nr 28 jest następnym blokiem pliku.

Przerywanymi liniami na rysunku zaznaczona te powiązania pomiędzy blokami, które wynikają z przedstawionego obok fragmentu tablicy FAT.

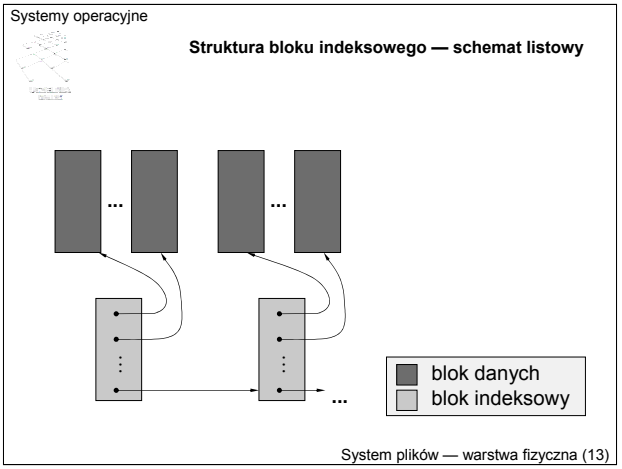


Atrybut *lokalizacja* w przypadku przydziału indeksowego musi przede wszystkim identyfikować blok indeksowy. Blok ten w całości wypełniony jest indeksami bloków z danymi. W przykładzie dla pierwszego pliku (zielonego) blok nr 3, jako blok indeksowy zawiera numery bloków 1, 8, 27 i 28, jako bloków z danymi. Podobnie w przypadku pozostałych plików, przy czym blok indeksowy pliku drugiego (czerwonego) identyfikuje tylko 2 bloki.

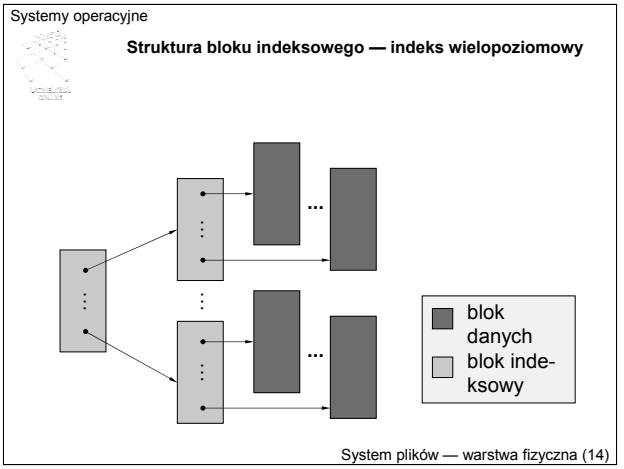


Blok indeksowy jest jednym z bloków przydzielanych w ramach zarządzania przestrzenią dyskową. Jego rozmiar jest skończony, w związku z czym skończona jest również liczba indeksów, którą można w nim przechować. Na przykład przy bloku 1KB i indeksie 4-bajtowym, w bloku można przechować 256 indeksów.

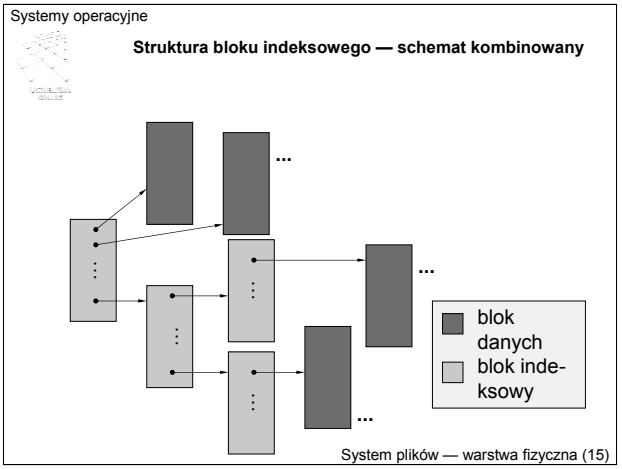
Jeśli liczba indeksów w jednym bloku nie jest wystarczająca, przydzielany jest kolejny blok indeksowy. Bloki indeksowe pliku należy jakoś powiązać. Można je powiązać w listę (schemat listowy), w drzewo (schemat wielopoziomowy) lub przyjąć rozwiązanie łączące kilka drzew o różnych głębokościach (schemat kombinowany).



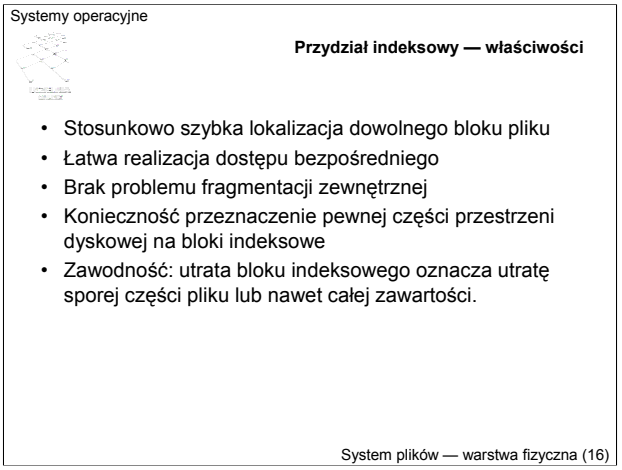
W schemacie listowym ostatni wskaźnik (indeks) w blok indeksowym wskazuje na następny blok indeksowy dla danego pliku. Jeśli zatem blok indeksowy może pomieścić 256 wskaźników, to 255 z nich odnosi się do bloków z danymi, a 1 do następnego bloku indeksowego.



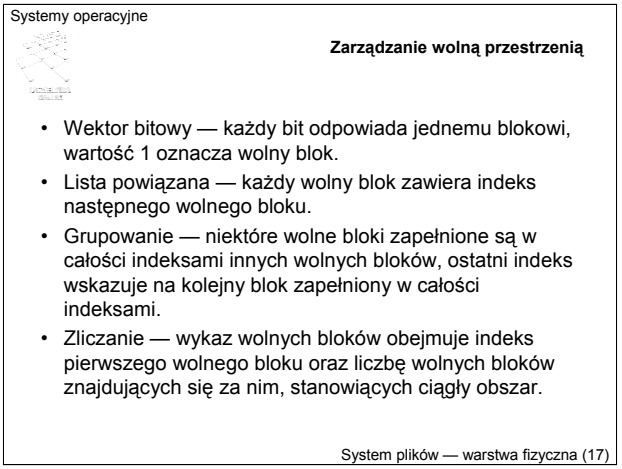
W schemacie wielopoziomowym na pierwszym poziomie jest jeden blok indeksowy, a każdy wskaźnik w tym bloku określa następny blok indeksowy itd. Dopiero ostatni poziom zawiera wskaźniki do bloków z danymi. W przypadku 256 wskaźników w jednym bloku oraz indeksie 2-poziomowym można zaadresować 256^2=65536 bloków z danymi, a przy indeksie 3-poziomowym 256^3=16777216 bloków z danymi.



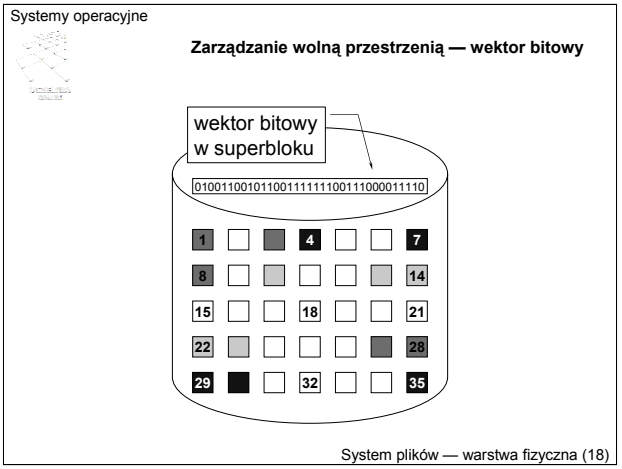
W schemacie kombinowanym część wskaźników w głównym bloku indeksowym określa bezpośrednio bloki z danymi, część wskazuje na kolejny blok indeksowy, adresujący bloki danych (indeks 2-poziomowy), część jest początkiem indeksu 3-poziomowego.



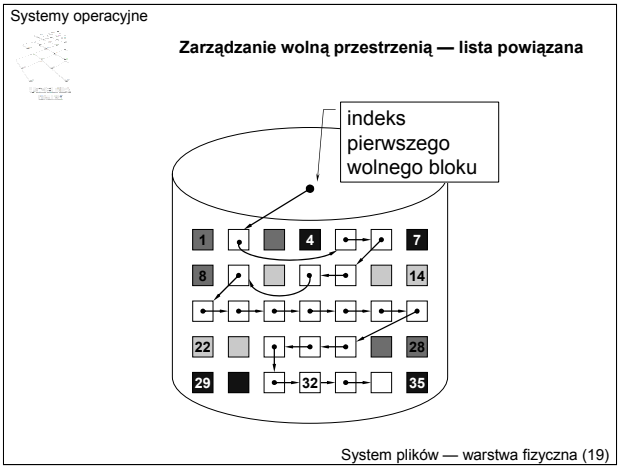
Podejście indeksowe umożliwia szybszą lokalizację bloku pliku, zawierającego wskazany fragment, niż w przypadku przydziału listowego. Przydział ten sprawdza się zatem zarówno przy dostępie sekwencyjnym jak i bezpośrednim. Koszt i ryzyko są podobne, jak w przypadku przydziału listowego.



Zarządzanie wolną przestrzenią sprowadza się do identyfikacji bloków, które nie są przydzielone i mogą zostać wykorzystane dla nowo tworzonych lub rozszerzanych plików. Wolne bloki można wykorzystać do tymczasowego przechowywania danych na potrzeby zarządzania. Dlatego część stosowanych rozwiązań przypomina przydział bloków na potrzeby plików. Można więc powiedzieć, że w tych podejściach wolna przestrzeń jest plikiem składającym się z wolnych bloków.



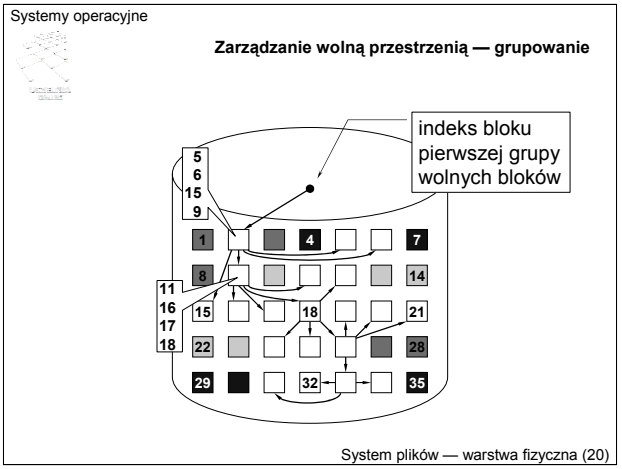
W wektorze bitowym każdy blok dyskowy (jednostka alokacji) reprezentowany jest przez jeden bit. Wartość 1 tego bitu oznacza, że dany blok jest wolny (można ewentualnie przyjąć odwrotną logikę). Efektywna implementacja takiego podejścia wymaga dostępności odpowiednich rozkazów manipulowania bitami w procesorze.



Lista powiązana tworzy z wolnych bloków plik zgodnie z koncepcją przydziału listowego. Powiązanie wolnych bloków polega więc na tym, że w bloku poprzednim znajduje się indeks bloku następnego, a indeks pierwszego bloku znajduje się w specjalnym miejscu w systemie plików.

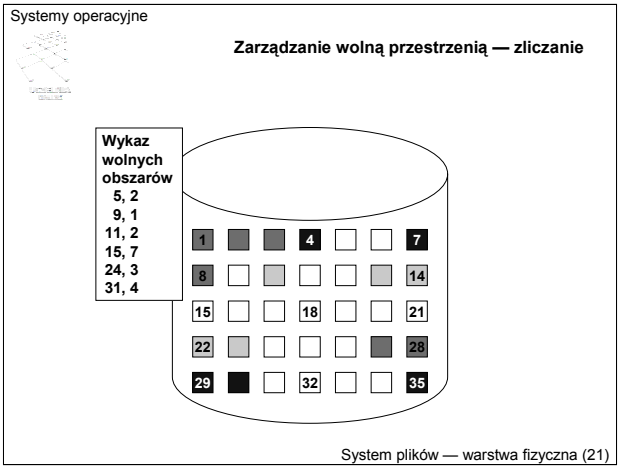
Przydział wolnego bloku polega na tym, że przydzielany jest blok, wskazywany jako pierwszy w superbloku, a indeks pierwszego bloku w superbloku zmieniany jest zgodnie z wpisem w bloku przydzielonym. W efekcie wskazuje zatem łańcuch, który zaczyna się od następnego wolnego bloku.

W przykładzie blok nr 2 zawiera indeks bloku 5, ten z kolei indeks bloku 6 itd.



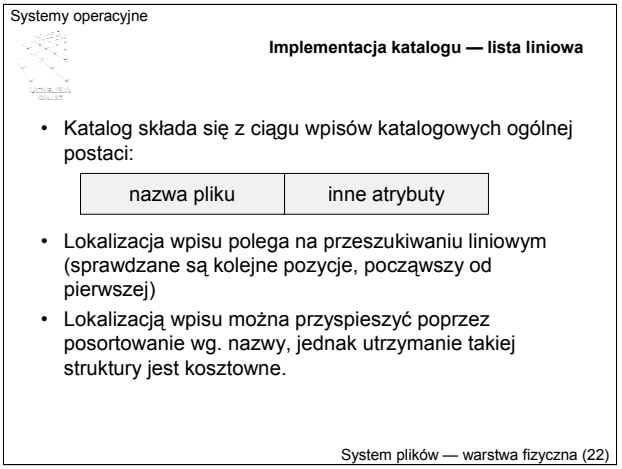
Grupowanie jest odpowiednikiem przydziału indeksowego z listową strukturą bloku indeksowego. Pierwszy wolny blok, wskazany w superbloku, zawiera indeksy *n* innych wolnych bloków, z których *n*-1 dotyczy bloków do natychmiastowej alokacji, a *n*-ty blok zawiera indeks następnego bloku grupującego *n*-1 indeksów kolejnych wolnych bloków oraz indeks następnego bloku grupującego.

W przedstawionym przykładzie blok nr 2 grupuje indeksy bloków 5, 6, 15, 9, przy czym blok nr 9 jest kolejnym blokiem grupującym.



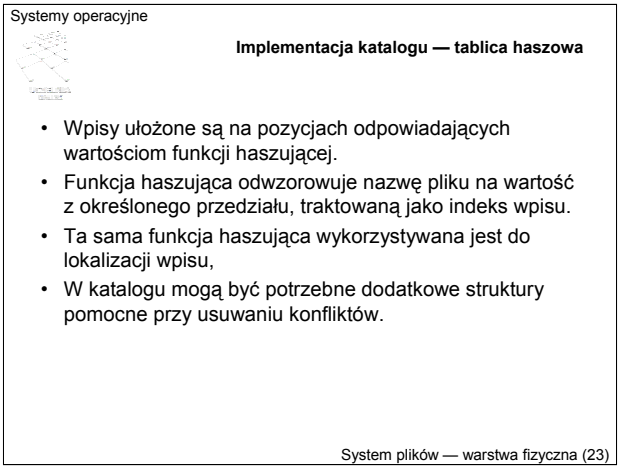
Zliczanie jest odpowiednikiem przydziału ciągłego. W przypadku kilku kolejnych (przylegających do siebie) wolnych bloków pamiętany jest tylko indeks pierwszego z nich oraz liczba wolnych bloków znajdujących się bezpośrednio za nim. Wykaz wolnych obszarów jest ciągiem wpisów, składających się z indeksu bloku oraz licznika. Zysk z tego podejścia ujawnia się, gdy występują duże ciągłe obszary wolne, np. wolna przestrzeń zaczynająca się od bloku 15.

Podejście taki mogłoby być połączone np. z listą powiązaną lub grupowaniem.

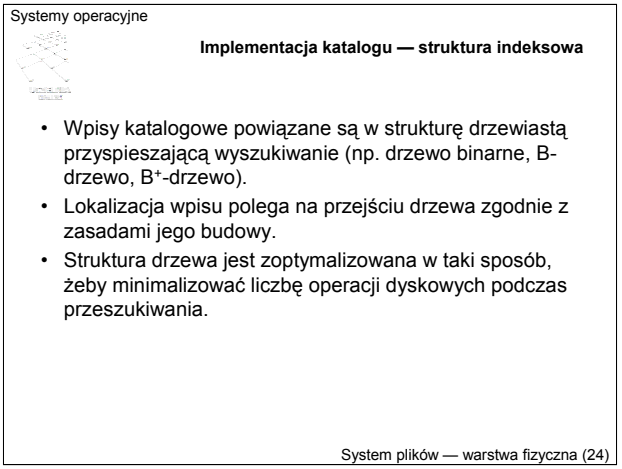


Katalog jest ciągiem wpisów, obejmującym przed wszystkim nazwę, gdyż przeszukiwanie katalogu odbywa się najczęściej po nazwie. Poza nazwą w katalogu mogą być inne atrybuty lub informacje o lokalizacji odpowiedniego obiektu opisującego plik.

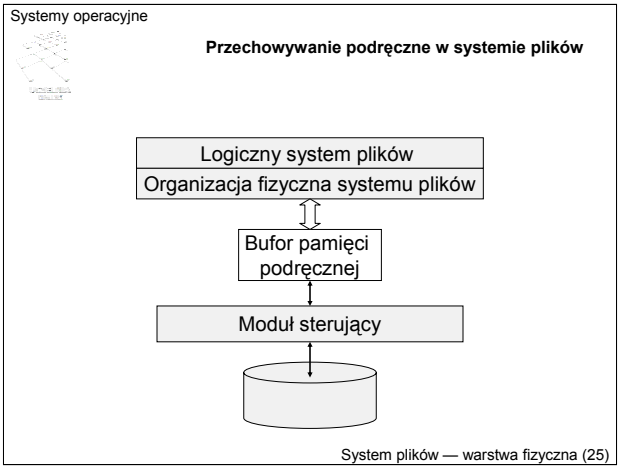
Lista liniowa jest najprostszą formą implementacji katalogu. Jest to po prostu ciąg wpisów. Przeszukiwanie takiego katalogu odbywa się w czasie liniowo zależnym od liczby wpisów. W przypadku małych katalogów rzeczywisty czas jest na tyle krótki, że jakakolwiek optymalizacja nie jest potrzebna. W przypadku większych katalogów korzyść z formy haszowej lub indeksowej może okazać się znacząca.



Pozycja danego wpisu wyznaczana jest zgodnie z wartością funkcji haszującej dla nazwy pliku. Działanie przykładowej funkcji haszującej mogłoby polegać na zsumowaniu kodów znaków tworzących nazwę, a następnie wyznaczeniu wartości *modulo liczba pozycji* z tej sumy. Wykorzystując tę samą funkcję przy wyszukiwaniu wpisu, można go szybko zlokalizować. Funkcja haszująca nie gwarantuje unikalności (nie jest to funkcja różnowartościowa), należy się więc liczyć z konfliktami, gdy ta sama wartość zostanie wyznaczona dla dwóch różnych nazw. W zależności od sposobu rozwiązywania konfliktów, potrzebne mogą być dodatkowe struktury.



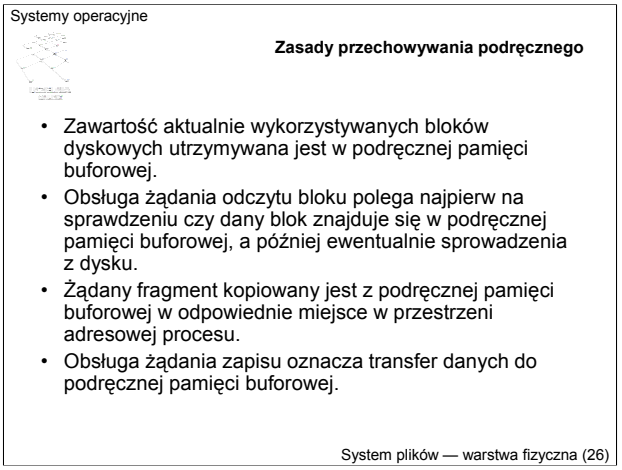
Inną formą przyspieszania lokalizacji wpisu jest struktura drzewiasta, oparta np. na B/B^+-drzewie. Struktura drzewiasta w zakresie czasu wyszukiwania daje efekt podobny jak posortowanie, jest przy tym łatwiejsza w aktualizacji. Wierzchołki w B/B^+-drzewie kojarzone są z blokami dyskowymi, co umożliwia optymalizację transferu danych pomiędzy jednostką centralną a urządzeniem przy dostępie do indeksu.



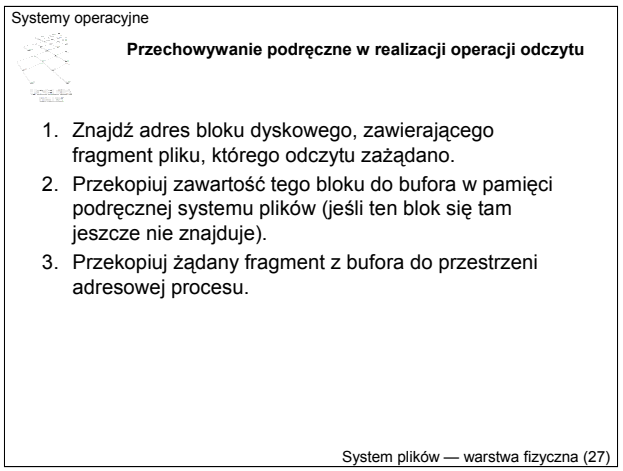
Operacja dostępu do danych w pliku wymaga ich sprowadzenia z urządzenia do pamięci operacyjnej, gdzie można je testować, zmieniać, po czym zażądać ponownego ich zapisania na urządzeniu.

Czytanie i pisanie bezpośrednio z / na dysk podczas wszystkich operacji dostępu do plików jest nieefektywne ze względu na czas reakcji dysku oraz relatywnie małą szybkość transmisji dyskowych.

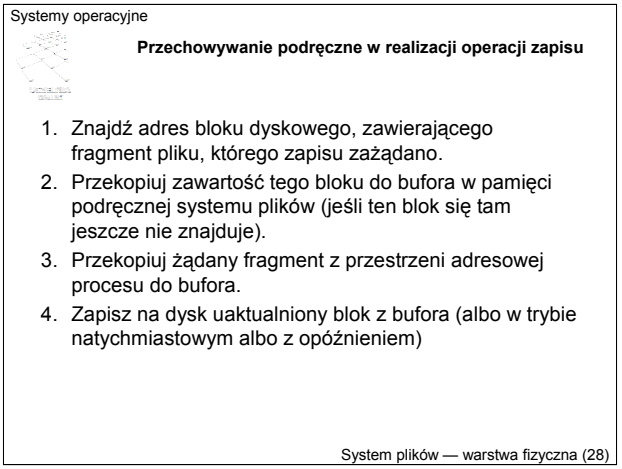
Minimalizacja dostępów do dysku możliwa jest przez utrzymywanie puli wewnętrznych buforów, zwanych podręczną pamięcią buforową (ang. buffer cache), które zawierają dane z ostatnio używanych bloków dyskowych.



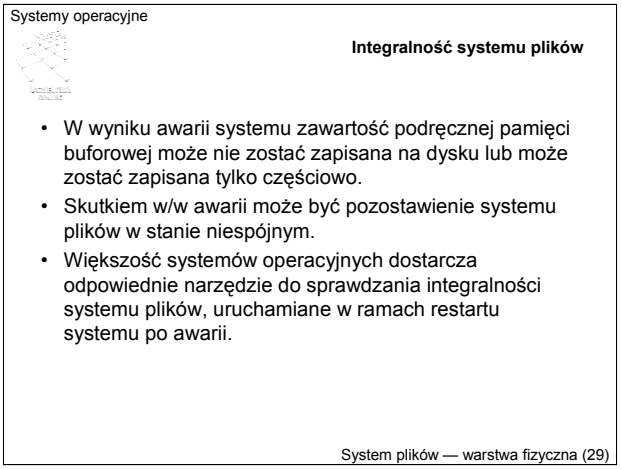
Dane w jednym buforze odpowiadają danym z jednego bloku dyskowego. Realizacja dostępu wymaga sprawdzenia dostępności bloku w podręcznej pamięci buforowej. Jeśli blok się tam znajduje, nie ma potrzeby wykonywania operacji dyskowej. W przeciwnym przypadku następuje sprowadzenie żądanego bloku z urządzenia do bufora. Sprowadzenie takie może oznaczać usunięcie (zastąpienie) dotychczasowej zawartości bufora. Jeśli zawartość ta była modyfikowana, wcześniej musi nastąpić zapis usuwanych danych we właściwym bloku dyskowym. Blok, dostępny w buforze, może być czytany oraz zapisywany, zależnie od praw dostępu do pliku, trybu otwarcia i rodzaju wykonywanych operacji. Modyfikacja zawartości bufora wymaga ponownego zapisu zawartości na dysku. Zapis taki może być wykonany natychmiast lub w późniejszym czasie, zależnie od trybu otwarcia. Póki nie nastąpi zwolnienie bufora na potrzeby nowo sprowadzanych danych, jego zawartość może być wykorzystywana przy kolejnych operacjach dostępu, gdyż bufor ten zawiera najświeższe dane. Konieczność natychmiastowego zapisu bloku dyskowego może być podyktowana bezpieczeństwem lub względami spójności. Niebezpieczeństwo wynika z możliwości utraty zawartości podręcznej pamięci buforowej w przypadku awarii komputera, zaniku zasilania itp. Problem spójności może powstać, jeśli do dysku jest dostęp bez pośrednictwa systemu operacyjnego, który buforuje dane. Przypadek taki może dotyczyć dysków w sieciowych systemach plików.



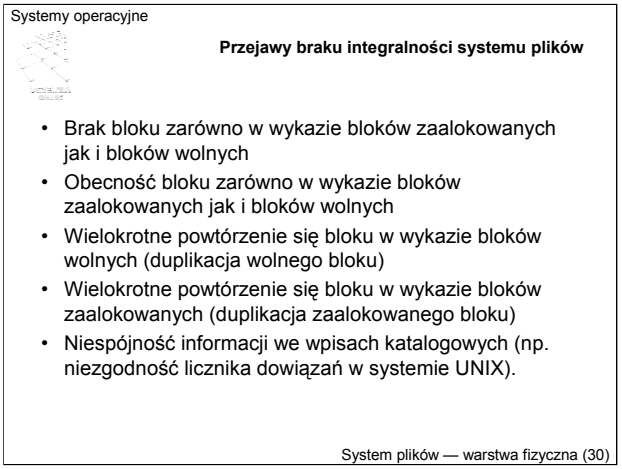
Schemat realizacji dostępu do odczytu w uproszeniu wygląda tak, że najpierw lokalizowany jest blok (lub bloki), zawierający czytany fragment. Sama lokalizacja w zależności od metody przydziały bloków może wymagać odczytu dodatkowych bloków, które też zostaną umieszczone w podręcznej pamięci buforowej. Po zlokalizowaniu bloku sprawdza się, czy znajduje się on w buforowej pamięci podręcznej. Jeśli nie ma go w pamięci podręcznej, następuje odczyt z urządzenia i umieszczenie zawartości w odpowiednim buforze. Ostatni etap polega na kopiowaniu danych z bufora pamięci podręcznej pod adres w przestrzeni procesu aplikacyjnego, wskazany w parametrach operacji.



Schemat postępowania jest podobny jak przy odczycie. Warto zwrócić uwagę, że dostęp do zapisu oznacza konieczność wcześniejszego odczytania bloku, ponieważ operacja zapisu może dotyczyć tylko fragmentu tego bloku.



Problem integralności może powstać wówczas, gdy w wyniku operacji następuje zmiana kilku bloków. Rozważmy przykład systemu plików z przydziałem indeksowym, w którym następuje rozszerzenie pliku. Rozszerzenie oznacza dołączenie kolejnego bloku danych i tym samym zmiany w plikach indeksowych oraz w identyfikacji bloków wolnych. Gdyby żadna z tych zmian nie została utrwalona, można przyjąć, że ze względu na awarię operacja nie udała się. Jeśli jednak tylko część tych zmian zostanie utrwalona na dysku, może się okazać, że blok jest zarówno w wykazie wolnych jak i przydzielonych jednostek lub też nie ma go w żadnym z tych wykazów.

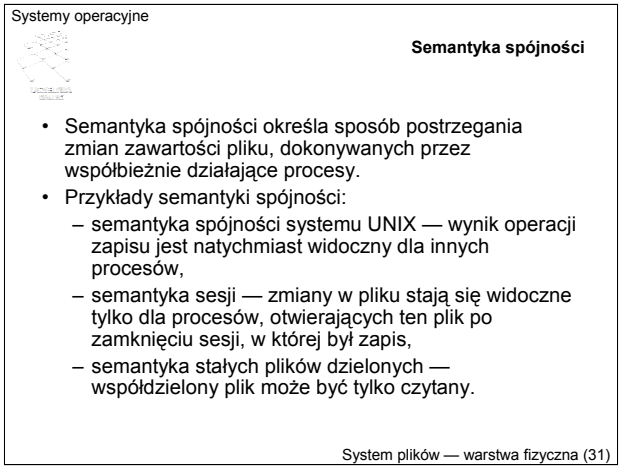


Pierwszy z przejawów braku integralności nie jest szczególnie niebezpieczny dla stanu systemu plików. Oznacza on po prostu tymczasową (do momentu wykrycia i usunięcia) utratę bloku. Usunięcie niespójności polega po prostu na dołączeniu bloku do wykazu bloków wolnych.

Poważniejsze w skutkach mogą być następne dwa przypadki. Obecność bloku w wykazie bloków wolnych jak i zajętych sama w sobie nie jest jeszcze problem i może być łatwo naprawiona przez usunięcie bloku z wykazu bloków wolnych. Jednak w wyniku żądania przydziału, błędnie identyfikowany blok może zostać zaalokowany ponownie i dochodzimy do kolejnego przejawu niespójności — duplikacji przydzielonego bloku.

Podobnie wygląda przypadek wielokrotnego powtórzenia się bloku w wykazie bloków wolnych. Jest ryzyko, że ten sam blok zostanie wielokrotnie przydzielony, ale wczesne wykrycie i usunięcie duplikacji nie powoduje żadnych dalszych konsekwencji. Warto zwrócić uwagę, że ten przejaw niespójności nie może się zdarzyć w przypadku użycia wektora bitowego do zarządzania wolną przestrzenią.

Poważniejsze w skutkach jest wielokrotne powtórzenie bloku w wykazie bloków przydzielonych plikom (tzw. skrzyżowanie), bo dostęp do każdego z plików (czy różnych fragmentów tego samego pliku) może spowodować niezależne modyfikacje. Próbą rozwiązania problemu może być powielenie bloku i przydzielenie w każdym miejscu wystąpienia osobnej kopii. Może się jednak okazać, że nie będzie to właściwa kopia w żadnym z tych miejsc.

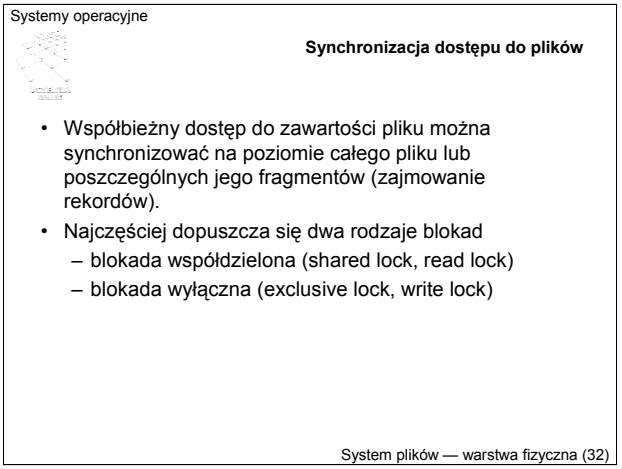


Semantyka spójności staje się istotna w przypadku współbieżnego dostępu do pliku wielu procesów, przy czym skutkiem dostępu są modyfikacje pliku. Semantyka spójności określa więc, w jaki sposób takie zmiany wpływają na obraz pliku w innych procesach.

Zgodnie z semantyką spójności systemu UNIX wszelkie zmiany są natychmiast dostępne dla następnych operacji odczytu, niezależnie od procesu. Wszystkie procesu korzystają po prostu z tej samej podręcznej pamięci buforowej.

Semantyka sesji oznacza, że wraz z otwarciem pliku tworzona jest sesja dostępu i wszelkie zmiany pliku w sesji staną się dostępne dopiero po jej zamknięciu. Jeśli zatem proces w czasie otwarcia sesji przez inny proces, otworzy swoją sesję, uzyska lokalną wersję pliku, która nie będzie uwzględniać dokonanych zmian w sesji współbieżnej. Dopiero otwarcie sesji po jej zamknięciu przez inny proces umożliwia dostęp do zmian.

Semantyka stałych plików dzielonych oznacza uniknięcie problemu, raczej niż jego rozwiązanie.



Zachowanie poprawnej struktury pliku przy współbieżnym realizowaniu operacji wymaga często koordynacji działań procesów, które mają dostęp do tego pliku. Koordynację taką w najprostszym przypadku można uzyskać poprzez wyłączność dostępu do pliku na czas wykonywania operacji. Blokada wyłączna jest często nadmiarowa, podobnie jak blokowanie całego pliku.

Wyróżnia się zatem blokadę

- współdzieloną, zakładaną na czas odczytu i dopuszczająca inne blokady współdzielone,
- wyłączną, zakładaną na czas modyfikacji i wykluczającą jakiekolwiek inne blokady.

Inną kwestią jest ziarnistość blokad. Bardzo często wystarczy zablokować tylko modyfikowany lub czytany fragment pliku.

Więcej szczegółów związanych z synchronizacją znajduje się w następnych modułach.

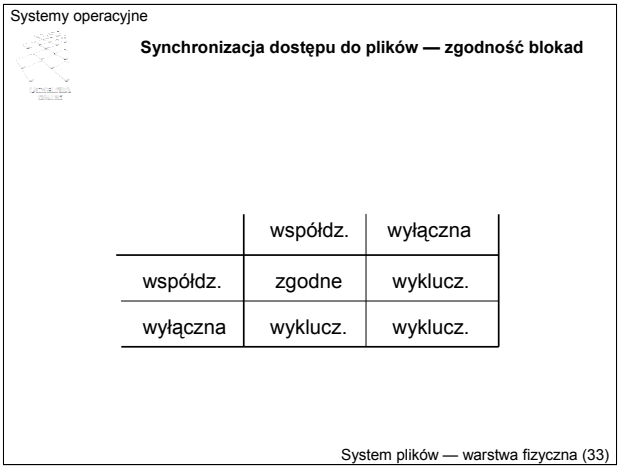


Tabela pokazuje, które rodzaje blokad mogą współistnieć, a które się wykluczają. Jak już wcześniej wspomniano, blokada do odczytu może współistnieć z inna blokadą do odczytu — współbieżny odczyt jest dopuszczalny. Żeby natomiast uniknąć odczytu danych częściowo tylko zmodyfikowanych lub nie dopuścić do hazardu w przypadku współbieżnych modyfikacji (patrz następny moduł) blokada wyłączna wyklucza inne blokady.