10. SYSTEM PLIKÓW

10.1. Pliki i katalogi

Plik to logiczna jednostka danych, definiowana przez SO, niezależnie od cech urządzeń.

System operacyjny udostępnia funkcje systemowe operujące na plikach.

SO odwzorowuje pliki na urządzeniach fizycznych.

Atrybuty plików: Nazwa, Typ, Położenie, Rozmiar, Ochrona, Czas, Data, Ident. użytkownika.

SO tworzy **Tablicę Otwartych Plików**, w której plik identyfikuje się poprzez **indeks** Tablicy.

➡ Po **zamknięciu** pliku przez proces, SO usuwa wpis z **T**ablicy **O**twartych **P**lików.

Plik może być otwarty przez kilku użytkowników jednocześnie.

Tablica Otwartych Plików zawiera **licznik otwarć** dla **każdego** pliku, pokazujący w ilu procesach dany plik został otwarty.

Każda operacja ***Zamknij*** zmniejsza **licznik otwarć**.

SO może zawierać **dwa** poziomy tablic plikowych.

Tablica Procesowa dotyczy plików otwartych w procesie,

zawiera informacje o sposobie korzystania z plików przez proces (wskaźnik dla READ/WRITE, prawa dostępu, itp).

Tablica Ogólnosystemowa Plików zawiera informacje niezależne od procesów (położenie pliku na dysku, jego wielkość, daty dostępu).

➔ Otworzono plik w jednym procesie:

wykonanie funkcji ***Otwórz*** w innym procesie **doda** nowy wpis do **TablicyProcesowej**,

z nowym wskaźnikiem bieżącym pliku

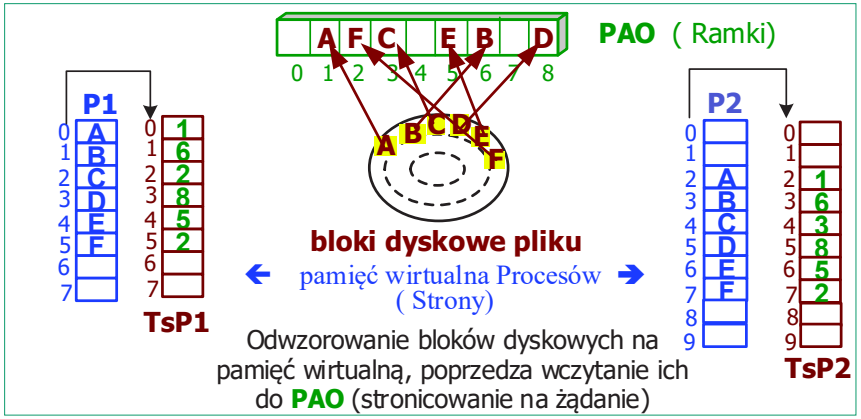
i wskaźnikiem do odpowiedniej pozycji w ***Tablicy Ogólnosystemowej***.

Blokowanie *(locking)* części otwartego pliku umożliwia (niektóre systemy):

- użytkowanie fragmentów pliku przez kilka procesów (poprzez strony dzielone),
- odwzorowywanie części pliku w PAO w systemach pamięci wirtualnej.

Odwzorowanie pliku w **PAO** *(memory mapping)* przyporządkowuje logicznie część wirtualnej przestrzeni adresowej Procesu do części **pliku**.

Operacje czytania i zapisywania TAKIEGO obszaru **PAO** są równoważne operacjom Czytaj/Zapisz do **pliku.**



Zamknięcie pliku odwzorowanego: dane odwzorowane w PAO zostają zapisane na dysku i usunięte z pamięci wirtualnej procesu.

Pisanie* do *pliku odwzorowanego: zmienia dane w pamięci wirtualnej;
Zmiany widoczne są dla innych procesów, odwzorowujących ten sam fragment pliku.

Wiele procesów może odwzorować ten sam plik w swoich pamięciach wirtualnych
➔ dzielenia danych.

➡Należy informować system operacyjny, jakiego **typu** plik został stworzony.
Można to zrealizować, włączając **typ** do nazwy pliku, jako jej części.

➡SO powinien odróżniać plik **źródłowy** od ***wynikowego***, sprawdzać czas ostatniej zmiany lub utworzenia oraz określać język programu źródłowego.
Wówczas wykonanie programu, którego plik źródłowy zmieniono po utworzeniu pliku **.exe** może wywołać automatyczną kompilację pliku źródłowego.

➡SO może wymagać, aby plik wykonywalny miał strukturę umożliwiającą systemowi:
-określenie miejsca PAO, gdzie należy plik umieścić,
-identyfikację komórki, w której znajduje się pierwszy rozkaz.

W Apple Macintosh każdy plik ma typ i atrybut swojego twórcy, zawierający nazwę programu, który go utworzył. Atrybut określa SO podczas tworzenia pliku.

Struktura plików w *Macintosh Operating System* jest dwuczęściowa:

-odnoga zasobu *(resource fork)* zawiera informacje dla użytkownika, np. etykiety przycisków wyświetlanych przez program; można je edytować i modyfikować.
Macintosh zawiera środki do wykonywania zmian danych w ***odnodze zasobu***.

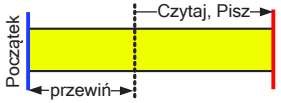
-odnoga danych*(data fork)*: kod programu lub dane - treść pliku w tradycyjnym rozumieniu.

❒ **Dostęp sekwencyjny** *(sequential access)*

Informacje w pliku przetwarzane są kolejno, rekord za rekordem.

Operacja **Czytaj** pobiera porcję pliku i automatycznie przesuwa do przodu wskaźnik położenia w pliku.

Operacja **Pisz** umieszcza dane na końcu pliku i ustawia wskaźnik za nowo zapisanymi danymi.



❒ **Dostęp bezpośredni** *(direct access)*

Plik jest ciągiem **ponumerowanych bloków**, zwanych też rekordami.

☛ Operacje plikowe muszą zawierać, jako parametr numer bloku.

Numer bloku przekazywany do SO jest zazwyczaj **numerem względnym bloku** *(relative block number), czyli* indeksem względem **początku pliku** (blok **0**, blok **1**, itd.)

Dla znanej długości ***L*** rekordu logicznego, zamówienie na **N**-ty rekord jest przekształcane na zamówienie ***L*** bajtów od adresu ***L*** * (**N - 1**) wewnątrz pliku.

❒ **Dostęp poprzez indeks**

INDEKS zawiera **wskaźniki** do **bloków** w **pliku**.

Przeszukuje się najpierw INDEKS, następnie poprzez wskaźnik dociera się do szukanego bloku w pliku i dalej do elementu → dalej szukanie liniowe

Indeksowanie umożliwia efektywne przeszukiwanie wielkich plików.

Dla dużych plików sam plik Indeksowy może być za duży, aby trzymać go w PAO.

Można utworzyć **dwa pliki Indeksowe**.

Pierwotny plik Indeksowy zawiera wskaźniki do **Wtórnego** pliku Indeksowego, który wskazuje na rzeczywiste bloki danych.

➔ Plik przechowywany jest w postaci **posortowanej** według zdefiniowanego klucza.

Najpierw przeszukuje się binarnie **IndeksGłówny**, otrzymując numer bloku **IndeksuWtórnego**.

Blok **IndeksuWtórnego** przeszukuje binarnie, aby odnaleźć blok z wymaganym rekordem ⇒ który jest już przeszukiwany ***liniowo***.

10.1.2. Struktura katalogowa

Zarządzanie plikami organizowane jest zazwyczaj w dwu częściach.

1. System plików podzielony jest na ***strefy***, nazywane ***tomami*** *(volumes).*

   Każdy tom zawiera, co najmniej jedną strefę, mieszczącą pliki i katalogi.

   Mogą wystąpić strefy, jako logiczne struktury kilku dysków.

2. Strefa zawiera informacje o zawartych w niej plikach.

   Informacje przechowywane są w pozycjach **katalogu urządzenia** *(device directory),* zawierającego dla każdego pliku danej strefy: nazwę, położenie, rozmiar, typ.

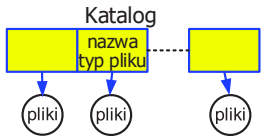
❒ **Katalog jednopoziomowy**

Wszystkie pliki są ujęte w tym samym katalogu.

Łatwo utworzyć i obsługiwać.

Pliki w tym samym katalogu, muszą mieć jednoznaczne nazwy.

Tworzenia plików o jednoznacznych nazwach staje się trudne przy wzroście liczby plików.



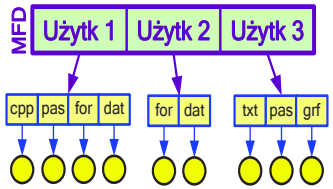
❒ **Katalog dwupoziomowy**

Po zarejestrowaniu użytkownika w systemie następuje przeglądanie **Głównego Katalogu Plików** (***M**aster **F**ile **D**irectory*- **MFD**), który jest indeksowany nazwami użytkowników lub numerami kont.

Użytkownik ma własny **Katalog Plików Użytkownika** (***U**ser **F**ile **D**irectory* - **UFD**).

Gdy użytkownik odnosi się do określonego pliku, wówczas przeszukuje się tylko jego własny katalog.

Odizolowanie wzajemne użytkowników nie zawsze jest pożądane.



◗Problem plików systemowych.

Wiele interpreterów poleceń traktuje nazwę ***polecenia*** daną SO, jako nazwę pliku do wykonania.

Standardowe rozwiązanie określa **specjalny katalog** dla plików systemowych.

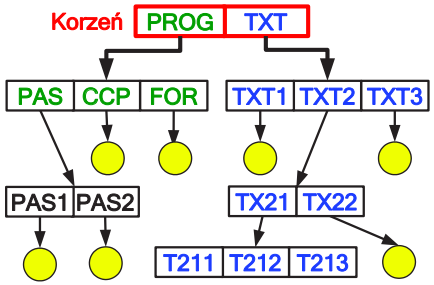
Gdy wystąpi nazwa pliku, SO przeszukuje najpierw katalog lokalny użytkownika.

Jeśli nie znajdzie w nim danego pliku, to automatycznie przeszukuje katalog systemowy.

❒ **Katalog drzewiasty**

System plików MS-DOS ma strukturę drzewa.

Drzewo rozpoczyna korzeń, czyli katalog główny.



Pliki to ***liście*** drzewa, do których wiodą jednoznaczne nazwy ścieżek idące od korzenia poprzez podkatalogi.

☛ Katalog jest **plikiem**, traktowanym specjalnie.

Jeden bit w każdym wpisie katalogowym określa, czy wpis dotyczy pliku (0) czy podkatalogu (1).

Tworzenie/Usuwanie katalogów realizują funkcje systemowe**.**

Bezwzględna nazwa ścieżki: droga od **korzenia** do określonego pliku; zawiera nazwy podkatalogów.

Względna nazwa ścieżki: **droga od bieżącego katalogu do określonego pliku**.

Decyzją polityczną jest sposób postępowania przy usuwaniu katalogu zawierającego wpisy.

Standardowo MS-DOS, nie usunie katalogu, jeśli nie jest on pusty.

Użytkownik musi najpierw usunąć wszystkie znajdujące się w nim pliki i podkatalogi.

UNIX umożliwia usunięcie katalogu, zawierającego podkatalogi z plikami.

Drzewiasty system katalogów umożliwia użytkownikom dostęp do plików innych użytkowników.

Użytkownik **A** może korzystać z plików użytkownika **B**, jeśli odwoła się do nazw ich ścieżki.

> W komputerze **Macintosh** zautomatyzowano wyszukiwanie programów wykonywalnych.
> System utrzymuje plik ***Desktop File***, zawierający nazwy i lokalizacje znanych programów wykonywalnych.
> Kiedy w systemie pojawi się nowy dysk lub połączenie z siecią, wtedy SO poszukuje na danym urządzeniu programów wykonywalnych.

❒ Acykliczny graf katalogów

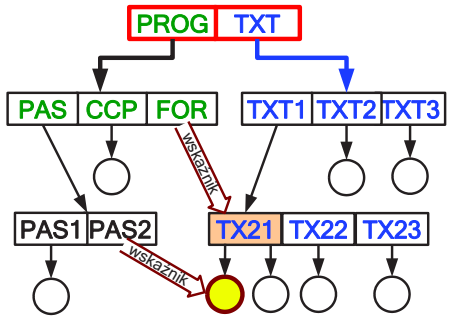
Struktura drzewiasta nie pozwala na dzielenie podkatalogów.

W grafie acykliczny ten sam podkatalog lub **plik** może występować w dwu różnych **katalogach**.

Plik utworzony przez jednego użytkownika pojawi się automatycznie we wszystkich podkatalogach dzielonych.

Graf acykliczny *(bez cyklu)* stanowi uogólnienie koncepcji drzewiastej struktury katalogu.

Pracując zespołowo można wszystkie pliki, które mają być dzielone umieścić w jednym podkatalogu.
Katalogi każdego z członków zespołu będą zawierały katalog plików dzielonych, jako podkatalog.



Implementacja plików i podkatalogów dzielonych można realizować w różny sposób.

Może to być ***dowiązanie*** *(link), będące* wskaźnikiem do innego pliku lub podkatalogu, co jest równoważne utworzeniem ***nowej*** pozycji w katalogu.

Dowiązaniem może być **bezwzględna** lub **względna** nazwy ścieżki (dowiązanie **symboliczne**).

- ☛ Wystąpienie odniesienia do pliku przeszukuje katalog.
- ☛ Pozycja w katalogu oznaczona jest, jako **dowiązanie**.

  Następuje przetłumaczenie **dowiązania**, czyli użycie pamiętanej w nim nazwy ścieżki do zlokalizowania rzeczywistego pliku.

  Dowiązania można **identyfikować** w katalogu na podstawie ich budowy (lub specjalnego typu danych w systemach operujących typami).

➡ SO ***pomija*** **dowiązania** podczas przeglądu drzew katalogowych, aby zachować strukturę bez cykli.

Jednemu plikowi może odpowiadać wiele bezwzględnych nazw ścieżek.

Do tego samego pliku mogą odnosić się różne nazwy plików.

Kiedy przestrzeń przydzieloną plikowi dzielonemu można mu odebrać?

Można usunąć plik, gdy ***ktokolwiek go usuwa***, lecz istnieje ryzyko pozostawienia wskaźników do już nieistniejącego pliku.

> Jeżeli **dzielenie** plików/katalogów zaimplementowano za pomocą dowiązań ***symbolicznych***, to usunięcie dowiązania nie narusza samego pliku
>
> – usuwane jest **tylko dowiązanie**.

◗ **Usunięcie wpisu pliku w katalogu,** zwalnia obszar pliku i **uwalnia dowiązanie**.

Można odnaleźć takie dowiązania i **usunąć je**, lecz przeszukiwanie to jest kosztowne, chyba, że wykaz dowiązań jest przechowywany z każdym plikiem.

Wolne **dowiązania** można **pozostawić** do **czasu**, aż **pojawi się próba ich użycia.**

Ponieważ plik o nazwie określonej przez dowiązanie nie istnieje, użycie dowiązania zostanie potraktowane jak **niedozwolona** nazwa pliku.

Co zrobić, gdy po usunięciu pliku zostanie utworzony nowy plik z tą samą nazwą, zanim ***zagospodaruje*** się **dowiązanie** do starego pliku ?

W systemie UNIX, Windows dowiązania symboliczne pozostają po usunięciu pliku i użytkownik musi być świadom, że pierwotny plik już nie istnieje lub został zastąpiony.

◗ **Plik zostaje usunięty wówczas, gdy** zostaną **usunięte** **wszystkie odniesienia** do **niego**.

- ☛ Należy umieć rozstrzygnąć czy zostało usunięte ostatnie odniesienie do pliku.

  Można prowadzić wykaz odniesień do pliku (wpisy katalogowe lub dowiązania symboliczne).

  Kiedy powstaje dowiązanie lub kopia wpisu katalogowego, wtedy dodaje się nową pozycję do wykazu odniesień do pliku.

  Usuwając dowiązanie lub wpis katalogowy, usuwa się też jego pozycję z wykazu.

◗ **Nie trzeba trzymać całych wykazów odniesień** - wystarczy **liczba odniesień** do **pliku**.

Nowe dowiązanie lub nowa pozycja w katalogu zwiększa ***licznik odniesień*** do pliku; usunięcie dowiązania lub pozycji z katalogu **zmniejsza** stan licznika.

Wyzerowanie ***licznika*** informuje, że nie ma już do danego pliku odniesień.

Metodę zastosowano w systemie UNIX dla ***dowiązań niesymbolicznych,*** przy czym ***licznik odniesień*** znajduje się w bloku informacyjnym pliku.

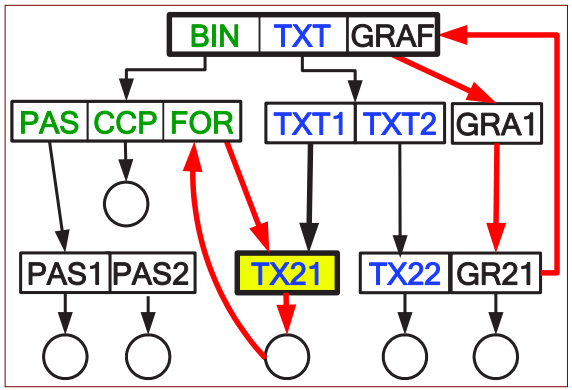
Nie wszystkie systemy zezwalają na dzielenie katalogów lub stosowanie dowiązań.

❒ **Graf ogólny katalogów**

Należy zagwarantować, że w acyklicznym grafie katalogów nie powstaną cykle.

Dowolne dodawanie dowiązań do istniejącej drzewiastej struktury katalogowej, może przekształcić ją w strukturę grafu z cyklami.

Jeśli przejrzano podkatalog dzielony i nie znaleziono w nim poszukiwanego pliku, to należy unikać jego powtórnego przeglądania.



Wystąpienie cykli w istniejących katalogach, może prowadzić do powstania niekończącej się pętli przeszukiwania.

Kiedy plik można usunąć?

Gdy **istnieją cykle,** możliwe jest, że **licznik odniesień** będzie **różny** od zera nawet wtedy, gdy **nie ma** już odniesienia do katalogu lub pliku.

Łączenie nieużytków *(garbate collection)* pozwala określić, kiedy usunięto ostatnie odniesienie i umożliwia ponowny przydział przestrzeni dyskowej.

❶ Wykonuje się obchód systemu plików i oznacza wszystko, do czego można dotrzeć.

❷ Niezaznaczone obszary wprowadza się do wykazu wolnych przestrzeni.

Łączenie nieużytków w systemie plików jest czasochłonne.

☛ Łączenie nieużytków uwarunkowane jest możliwością wystąpienia cykli w grafie.

☛ Trudno uniknąć cykli, gdy do struktury dodaje się **nowe dowiązania**.

Algorytmy wykrywania cykli w grafach, wymagają dużych nakładów obliczeniowych, dla grafów znajdujących się w pamięci dyskowej.

10.1.3. Ochrona

Ochrona przed: -fizycznym **uszkodzeniem** (niezawodność),
-niewłaściwym **dostępem** (ochrona).

Dostęp zależny od **tożsamości** kojarzy z każdym plikiem i katalogiem **wykaz dostępów** *(access list),* zawierający:
-nazwy użytkowników,
-dozwolone rodzaje dostępu.

Wadą wykazów dostępów jest ich długość, która jest wartością zmienną.

Aby skrócić **wykaz dostępów** do każdego z plików odnosi się trzy klasy użytkowników:

Właściciel: użytkownik, który utworzył dany plik.

Grupa albo **zespół:** zbiór użytkowników, wspólnie korzystających z pliku.

Wszyscy: wszyscy inni użytkownicy.

W systemie UNIX tworzeniem grup i ich uaktualnianiem zajmuje się tylko zarządca tej usługi systemowej (lub dowolny użytkownik o zwiększonych przywilejach).
Kontrola ta jest sprawowana na zasadzie współpracy systemu z człowiekiem.

Zdefiniowanie ochrony wymaga trzech pól.

Każde pole jest zbiorem bitów, z których każdy albo **pozwala**, albo **zabrania** określonego rodzaju dostępu.

W systemie UNIX zdefiniowano **trzy** pola **3-bitowe** ***rwx*:** **r** -czytanie,
w -pisanie,
x -wykonanie.

Osobne pola dla: -właściciela pliku,
-podlegającej właścicielowi grupy,
-dla reszty.

Inny sposób ochrony to **przypisanie każdemu plikowi hasła.**

Wielopoziomowe struktury katalogów wymagają mechanizmów ochrony katalogów.

W grafach acyklicznych lub ogólnych:
-pliki mogą mieć po kilka nazw ścieżek,
-prawa dostępu do pliku mogą się różnić między sobą w zależności od użytej przez użytkownika nazwy ścieżki.

W systemie UNIX ochrona katalogu jest podobna do ochrony pliku.

Z każdym podkatalogiem skojarzone są **trzy** pola: **Właściciela,**
Grupy
Wszyscy,
przy czym każde z nich zawiera 3 bity: ***rwx***.

Użytkownik może wyprowadzać zawartość katalogu, gdy ustawiono bit **r**.

10.1.4. Semantyka spójności *(cosistency semantics)*

Semantyka spójności jest kryterium oceny systemu plików realizującego **dzielenie plików**.

Określa warunki, przy których zmiany danych wykonywane przez jednego użytkownika obserwowalne są przez innych użytkowników.

Sesja plikowa -ciąg dostępów do pliku (Czytanie/Pisanie), zgłaszany przez użytkownika do **tego samego** pliku (zawarty między operacjami Otwierania i Zamykania pliku).

❒ **Semantyka spójności systemu UNIX**

1. Wynik operacji Pisania wykonany na otwartym pliku **jest widoczny natychmiast** dla innych użytkowników, którzy mają równocześnie otwarty ten sam plik.
2. Istnieje tryb, w którym użytkownicy ***wspólnie korzystają*** ze wskaźnika bieżącego położenia w pliku.
   Przesunięcie wskaźnika przez jednego użytkownika oddziałuje na wszystkich użytkowników dzielących plik.
   ☛ Plik ma **jeden obraz**, do którego odnoszą się wszystkie możliwe dostępy.

❒ **Semantyka spójności w systemie Andrew**

1. Wynik operacji Pisania wykonany na otwartym pliku **nie jest widoczny** natychmiast dla innych użytkowników, którzy mają równocześnie otwarty ten sam plik.
2. Zmiany wprowadzone do pliku będą widoczne po jego zamknięciu, czyli w następnych sesjach.
   → Otwarte, **bieżące** egzemplarze danego pliku **nie** odzwierciedlają tych zmian.

➔Plik w tej semantyce może być w tym samym czasie skojarzony z ***kilkoma* obrazami**.
Wówczas wielu użytkownikom może wykonywać współbieżnie na ich obrazach pliku operację Czytania i Pisania.

❒ **Semantyka stałych plików dzielonych** *(immutable shared files)*

Gdy twórca pliku zadeklaruje jego ***dzielenie***, plik przestaje podlegać dalszym zmianom.

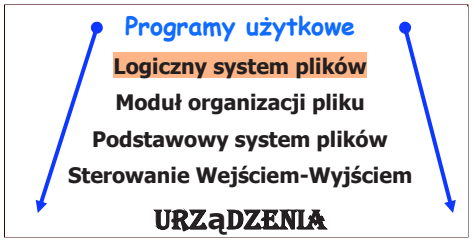
Właściwości pliku stałego: -nazwy pliku nie może być użyta ponownie,
-zawartość pliku nie może ulec zmianie.

10.3. Implementacja plików

Informacja między PAO a dyskiem przesyłana jest **blokami** (jeden lub więcej sektorów).

1. Jak użytkownik widzi system plików: definicja pliku, atrybuty, operacje, struktura katalogowa.

2. Algorytmy i struktury danych odwzorowujące logiczny system plików na urządzenia fizyczne.



W warstwowym systemie plików wyższe poziomy korzystają z właściwości niższych poziomów do wytworzenia nowych właściwości.

◗ **Sterowanie We/Wy** *(I/O control)* -składa się z ***modułów obsługi urządzeń*** i procedur obsługi przerwań, związanych z przesyłaniem informacji PAO a systemem dyskowym.

Moduł obsługi urządzenia *(device driver)* tłumaczy z We polecenia wysokiego poziomu (pobierz blok 123) i zwraca na Wy niskopoziomowe rozkazy dla sterownika sprzętowego.

◗ **Podstawowy system plików** (*basic file system)* wydaje ogólne instrukcje odpowiedniemu modułowi obsługi urządzenia w celu **czytania** i **pisania bloków** na dysku.

Fizyczny blok dysku określają wartości liczbowe: napęd, cylinder, powierzchnia, sektor.

◗ **Moduł organizacji pliku** *(file organization module)*:

-**tłumaczy** adresy **logiczne** bloków (ponumerowane od 0 do N) na adresy fizyczne.
Numery bloków fizycznych na ogół nie odpowiadają numerom logicznym.

-**zarządza wolnymi** obszarami (nieprzydzielone bloki) i udostępnia je na życzenie.

◗ **Logiczny system plików** *(logical file system)* używa **struktury katalogowej**.

Poprzez symboliczne nazwy plików dostarcza informacji modułowi organizacji pliku.

Zawiera mechanizmy ochrony i bezpieczeństwa pliku.

Strukturę katalogową obsługuje się **B**lok **K**ontrolny **P**liku –**FCB**.

PCB: zezwolenia dostępu, daty utworzenia i operacji, właściciel, rozmiar, lokalizacja bloku danych.

Program użytkowy wywołuje **logiczny** system plików, który zna format struktur katalogowych.

➔ Tworzenie nowego pliku:
-**przydzielenie** nowego **FCB,**
-**odczyt** katalogu do PAO i dodanie nowego wpisu,
-**zapisy** w **FCB**,
-**zapis** katalogu na dysk.

UNIX traktuje katalog tak samo jak plik, ustawiając ***pole typu*** na katalog.
WINDOWS realizuje osobne wywołania systemowe dla plików i katalogów.

Fragmenty struktury katalogowej przechowywane są często w pamięci podręcznej.

- Należy **Otworzyć** plik przed udostępnieniem go procedurom We/Wy.

  -**przeglądanie** struktury katalogowej w celu odnalezienia nazwy pliku,

  -kopiowane FCB pliku do **O**gólnosystemowej **T**ablicy **O**twartych **P**lików (w **PAO**).

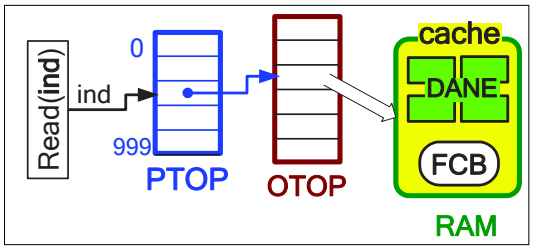
  -**wykonanie wpisu** do **P**rocesowe **T**ablicy **O**twartych **P**lików (wpis zawiera m.in. wskaźnik do Ogólnosystemowej Tablicy Otwartych Plików).

  -**udostępnienie** wskaźnika-**Indeksu** do odpowiedniej pozycji **P**rocesowej **T**ablicy **O**twartych **P**lików → dalsze odwołania wykonywane są za pomocą tego **Indeksu**.

  ☛ Nazwy **symboliczna** pliku staje się już **nieistotna**, gdyż wszystkie niezbędne informacje zawarte są w **FCB** (przekopiowanym z dysku).

Indeks to **deskryptor pliku** *(file descriptor)* lub **uchwyt pliku** *(handle).*

Po otwarciu pliku wszystko, z wyjątkiem samych bloków danych, znajduje się w PAO.



Uwaga: **Open** przeszukuje najpierw **O**gólnoSystemową**T**ablicę**O**twartych**P**lików, sprawdzając czy plik nie jest używany w innym Procesie.

Jeśli jest używany, to tworzy się wpis w **P**rocesowej**T**ablicy**O**twartych**P**lików z odwołaniem do **O**gólno**S**ystemowej**T**ablicy**O**twartych**P**lików.

- System plików należy **zamontować**, zanim stanie się dostępny dla procesów w SO.

  Proces podłączania dysku lub partycji do głównego systemu plików nazywamy **montowaniem**.

  Dyski lub partycje podłączone do **/** lub ****, widoczne są w katalogach zwanych **punktami montowania**.

  SO otrzymuje: -nazwę urządzenia,

  -miejsce w strukturze plików, do którego należy przyłączyć system plików (punkt montażu – *mount point*) np.: C:****DOS\Katalog\Podkalalog\PLIK

  Po „montażu" SO sprawdza czy urządzenie zawiera właściwy system plików.

  Prosi **moduł obsługi urządzenia** o przeczytanie katalogu urządzenia i sprawdza, czy katalog ma oczekiwany format.

  Na koniec SO ***zaznacza*** w strukturze katalogowej zamontowanie pliku.

/ (slash). - katalog główny systemu UNIX (root directory)

10.4. Przydział miejsca na dysku

Metody przydziału miejsca na dysku:

ciągły **listowy** **indeksowy**

10.4.1. Przydział ciągły *(contiguous allocation)*

Każdy plik musi zajmować ciąg kolejnych bloków na dysku.

Adresy dyskowe definiują na dysku uporządkowanie liniowe.

Jeżeli z dyskiem kontaktuje się tylko *jedno* zadanie, dostęp do bloku **b + 1** po bloku **b** nie wymaga na ogół ruchu głowicy (co najwyżej będzie to przesunięcie o jedną ścieżkę).

Przydział ciągły pliku określa:

-adres dyskowy,

-długość pierwszej porcji (w blokach dyskowych).

Katalog

Plik	Początek	Długość
File1	**0**	**4**
File2	**7**	**6**
File3	21	6
File4	31	8

Jeśli plik składa się z **n bloków** i zaczyna od adresu **bloku B**, to zajmuje bloki **B, B + 1, B + 2, ..., B + n - 1**.

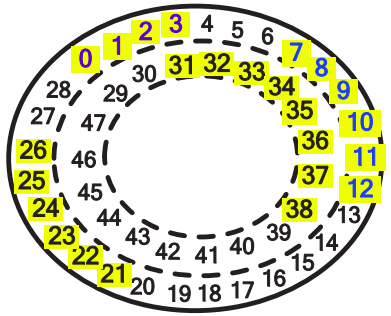
☛ Przy **dostępie sekwencyjnym** system plików pamięta adres dyskowy **ostatniego** bloku, do którego było odniesienie.

Bezpośredni dostępu do bloku **k** w pliku zaczynającym się od **bloku b** realizuje natychmiast operacja: **b** + **k**.

Trudność znalezienie miejsca na nowy plik.

Jest to problemu ***dynamicznego przydziału pamięci***.

Jak spełnieni zamówienie o rozmiarze **n** na podstawie listy wolnych dziur ?



Rys. 10.5. Przydział ciągły

Strategie **pierwszego** i **najlepszego dopasowania** są często stosowane do wybrania wolnego obszaru z listy dostępnych obszarów.

Nie są najlepsze ze względu na wykorzystanie pamięci, lecz metoda **pierwszego dopasowania** jest szybsza.

➧ Algorytm nie rozwiązuje problemu ***fragmentacji zewnętrznej***.

Fragmentacja zewnętrzna:, kiedy wolna przestrzeń nie jest ciągła.

◗ **Określenie wielkości obszaru potrzebnego dla pliku.**

Przy tworzeniu pliku musi być znaleziony i przydzielony **cały** wymagany przez niego obszar.

Skąd twórca ma znać rozmiar tworzonego pliku?

Jeśli plikowi przydzieli się za mało miejsca, to może być problem z rozszerzeniem.

Istnieją wtedy dwa rozwiązania:

Pierwsze: -zakończenie programu użytkownika z odpowiednim komunikatem błędu.
Użytkownik musi wówczas przydzielić więcej miejsca i wykonać program od nowa.

Drugie: -znalezienie większej ***dziury***, skopiowanie zawartości pliku do nowego obszaru i zwolnieniu poprzedniego.

Użytkownik nie musi być jawnie informowany o tym, co się zdarzyło;

System pracuje mimo trudności – aczkolwiek coraz wolniej.

➡ Plikowi **rosnącemu** powoli w długim okresie czasu, należy przydzielić obszar dostosowany do jego **końcowego** rozmiaru;
duża część tej przestrzeni może długo pozostawać niewykorzystana - *fragmentacja wewnętrzna.*

◗ **Modyfikacja metody przydziału ciągłego**:

-na początku przydziela się **kawałek ciągłego** obszaru,

-gdy zaczyna **GO** brakować, dodaje się następny - **nadmiarowy** kawałek.

Informacja o położeniu bloków pliku zawiera:

-adres pierwszego bloku,

-liczbę bloków,

-połączenie z 1-szym blokiem w obszarze nadmiarowym.

10.4.2. Przydział listowy *(linked allocation)*

Plik jest **listą** powiązanych bloków dyskowych, ulokowanych gdziekolwiek na dysku.

☛ Katalog zawiera **wskaźnik** do pierwszego i ostatniego bloku pliku.

Katalog

Plik	Początek	Koniec
File1	**3**	**20**
File2	21	8

* Każdy blok zawiera **wskaźnik** do następnego bloku.

-Wskaźniki te nie są dostępne dla użytkownika.

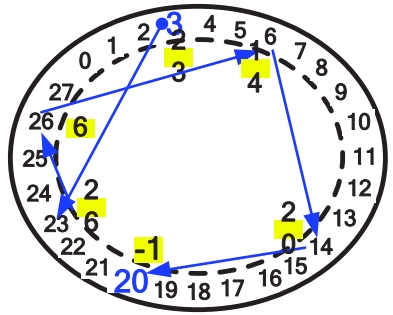
Jeśli blok ma 512B, a adres dyskowy (wskaźnik) zajmuje 4B, to dla użytkownika pozostaje 508 B do dyspozycji.

W celu utworzenia pliku tworzy się **nowy wpis** w **katalogu.**

Każda pozycja w katalogu ma wskaźnik do pierwszego dyskowego bloku pliku.

Początkowa wartość wskaźnika jest ***NULL*** w celu zaznaczenia, że plik jest pusty.

Pole rozmiaru przyjmuje wartość zero.



Rys. 10.6. Przydział listowy

Pisz uruchamia szukanie -przez system zarządzania wolnymi obszarami- nie zajętego bloku, następnie zapisuje go oraz dowiązanie do końca pliku.

Czytanie pliku odbywa się według wskaźników zapamiętanych w kolejnych blokach.

Nie ma zewnętrznej fragmentacji.

Nie trzeba znać rozmiaru pliku w chwili jego tworzenia.

Wady przydziału listowego:

➔ **Realizacja dostępu bezpośredniego** jest **niewydajna.**

Aby znaleźć **k-ty** blok pliku, należy zacząć od początku pliku i postępować za wskaźnikami aż dotrze się do bloku **k**.

☛ Każdy dostęp do wskaźnika wymaga czytania dysku.

➔ **Potrzebna dodatkowa przestrzeń zajmowana** przez **wskaźniki.**

Niedogodność tę można zredukować przez grupowanie kilku bloków w **klastery** *(clusters)* i przydzielanie klasterów zamiast bloków (wskaźnik na klaster).

Użycie klasterów może powodować wzrost ***wewnętrznej fragmentacji***.

➔ **Pliki powiązane** są **wskaźnikami rozrzuconymi** po **całym dysku.**

Co się stanie, gdyby jakiś wskaźnik ulegnie uszkodzeniu ?

Błąd w oprogramowaniu SO lub w urządzeniu dyskowym może pobrać zły wskaźnik i spowodować dowiązanie (pliku) do listy wolnych przestrzeni lub do innego pliku.

Pewnym rozwiązaniem jest zapamiętywanie w każdym bloku nazwy pliku i względnego numeru bloku lecz zwiększa to nakłady ponoszone na każdy plik.

10.4.3. Przydział indeksowy

W przydziale listowym wskaźniki do bloków rozrzucone są z blokami po całym dysku i mogą być osiągane po kolei (niewydajny dostęp bezpośredni)

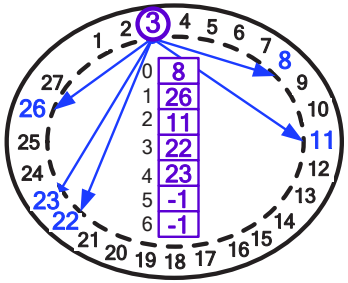
Przydział indeksowy skupia wskaźniki w **Bloku Indeksowym** *(index block).*

☛Każdy plik ma własny **Blok Indeksowy**, będący **Tablicą Adresów** bloków dyskowych.

Pozycja **k-ta** w bloku indeksowym wskazuje na **k-ty** blok pliku.

➡ Katalog zawiera **adres bloku indeksowego**.

Inicjalizacja pliku ustawia wskaźniki w bloku indeksowym na ***NULL***.



Rys. 10.7. Przydział indeksowy

Katalog

Plik	Blok indeksowy
File1	**3**

Odczytanie **k-go** bloku to zaadresowanie **Tablicy Adresów** indeksem **k**, i pobranie adresu odpowiedniego bliku.

Gdy blok **k** jest po raz pierwszy zapisywany:

-pobiera się go ze zbioru wolnych obszarów,

-jego adres zostaje umieszczony w **k-tej** pozycji **bloku indeksowego**.

Przydział indeksowy umożliwia dostęp bezpośredni bez zewnętrznej fragmentacji, gdyż zamówienie na dodatkowy obszar może spełnić wolny blok znajdujący się gdziekolwiek.

Wadą przydziału indeksowego **jest marnowanie przestrzeni.**

Wskaźniki bloku indeksowego zajmują na ogół więcej miejsca niż wskaźniki przy przydziale listowym.

Niech plik ma tylko jeden lub dwa bloki.

W przydziale listowym tracimy miejsce tylko dla jednego wskaźnika na blok (lub dwa).

W przydziale indeksowym trzeba przydzielić **cały blok** nawet wówczas, gdy tylko jeden lub dwa wskaźniki będą w nim różne od ***null***.

Jaka powinna być **wielkość bloku indeksowego ?**

Mały, gdyż każdy plik musi mieć blok indeksowy.

Duża, gdyż umożliwia pomieszczenie wszystkich wskaźników do wielkiego pliku.

- **Schemat listowy: blok indeksowy** mieści się w **jednym bloku dyskowym.**

  Można łączyć kilka bloków indeksowych.

  Blok indeksowy może zawierać: -nazwę pliku
  -sto pierwszych adresów bloków dyskowych.

  Następny adres (ostatnie słowo w bloku indeksowym) będzie miał wartość ***null*** (dla małego pliku) lub będzie wskaźnikiem do innego bloku indeksowego (dla dużego pliku).

- **Indeks wielopoziomowy: blok indeksowy 1-go** poziomu wskazuje zbiór bloków indeksowych **2-go** poziomu,
  -indeksy 2-go poziomu wskazują na bloki pliku.

  Indeks pierwszego poziomu wskazuje blok indeksowego drugiego poziomu, za pomocą którego odnajduje się potrzebny blok danych.

  Dany blok 4096B.
  W bloku indeksowym mamy 1024 czterobajtowych wskaźników.
  Dwa poziomy indeksów adresuje 1048576 bloków, co daje pliki o rozmiarach do 4 GB.

- **Schemat kombinowany:** przechowuje się np. pierwszych **18** wskaźników bloku indeksowego w i-węźle pliku.

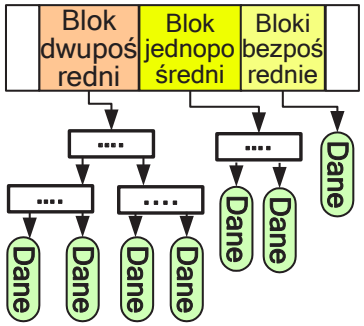
  Pierwszych **15** wskaźników wskazuje na ***bloki bezpośrednie***, tzn. zawierające adresy **bloków z danymi pliku**.

  Dzięki temu dane w małych plikach (nie dłuższych niż 15 bloków) nie muszą mieć oddzielnego bloku indeksowego.

  Dla rozmiaru bloku 4 KB, można osiągać bezpośrednio dane o wielkości do 60 KB.

  Następne **trzy** wskaźniki (przykładowo **16, 17, 18**) wskazują na **bloki pośrednie**.

  Pierwszy wskaźnik bloku pośredniego jest adresem ***bloku jednopośredniego***.



Blok ***jednopośredni*** zawiera **adresy** bloków z **danymi**.

Blok **dwupośredni** zawiera adres bloku, który zawiera adresy innych bloków, ze wskaźnikami na bloki danych.

Trzeci wskaźnik zawiera adres **bloku trójpośredniego**.

➔ **Bloki indeksowe mogą być przechowywane w pamięci CACHE, lecz bloki danych mogą być rozrzucone po całym dysku.**

10.5. Zarządzanie wolną przestrzenią dyskową

Należy zagospodarować przestrzenie po usuniętych plikach.

System utrzymuje listę wolnych obszarów dyskowych.

10.5.1. Wektor bitowy

Wektor bitowy implementuje listę wolnych obszarów.
+ Każdy blok reprezentuje **1 bit**.

Dla bloku **wolnego** bit ma wartość **1**.
Dla bloku **przydzielonego** dany bit wynosi **0**.

Niech bloki: **2**, **3**, **4**, **5**, **8**, 9, **10**, **11**, **12**, **13**, **17**, **18**, **25**, **26**, **27** będą wolne, a pozostałe przydzielone.

Wygląd mapy wolnych obszarów: **00111100 11111100 01100000 01110000 0**...

Metoda wydajnie odnajduje pierwszy **wolny** blok lub **n** kolejnych wolnych bloków na dysku.

Macintosh korzysta z wektora bitowego podczas przydzielania miejsca na dysku.

-sprawdza **kolejne słowa** w mapie bitowej czy są **równe 0**,
(wyzerowane słowo oznacza zbiór przydzielonych bloków).

-w pierwszym niezerowym słowie szuka pierwszego **bitu 1**, którego położenie wyznacza lokalizację pierwszego wolnego bloku.

Numer bloku określa wyrażenie:

Liczba**B**itów_w_**S**łowie • **L**iczba**S**łów**Z**erowych + **P**ozycja**P**ierwszego**Bitu 1**

Wydajność metody wektorów bitowych determinowana jest możliwością przechowywania wektora w PAO.

$$wektor\ bitowy[\text{bajt}] = \frac{pojemnosc\ dysku}{8 * blok\ plikowy}$$

Dla dysku **12GB** pracującego z blokami 512 bajtów, wektor bitowy zajmuje przestrzeń ok. **3MB**.

10.5.2. Lista wiązana

Można powiązać ze sobą wszystkie wolne bloki dyskowe i przechowywać wskaźnik do pierwszego wolnego bloku na dysku.

Pierwszy blok zawiera wskaźnik do ⇐ następnego wolnego bloku ⇐ itd.

Blok **2** zawiera wskaźnik do bloku **3**, który powinien zawierać wskaźnik do bloku **4**.

Metoda **nie** jest wydajna -przeglądanie listy wymaga odczytania każdego bloku, co jest kosztowne ze względu na operacje We/Wy.

☛ Przeglądanie listy wolnych bloków nie jest wykonywane często.

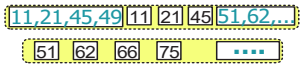
➧ Zazwyczaj SO potrzebuje wolnego bloku aby przydzielić go plikowi, więc pobiera pierwszy blok z listy bloków wolnych.

W **FAT** gospodarka wolnymi blokami włączona jest do struktury danych dotyczącej przydziału.

10.5.3. Grupowanie

Można przechowywać **adresy n** wolnych bloków w **pierwszym** wolnym bloku.

Pierwszych **n - 1** z tych bloków to bloki rzeczywiście wolne.



Ostatni blok zawiera **adresy** następnych **n** wolnych bloków itd.

Metoda skutecznie odnajduje adresy przy dużej liczbie wolnych bloków.

10.5.4. Zliczanie

Przylegające do siebie bloki można przydzielać i zwalniać jednocześnie, zwłaszcza wtedy, gdy stosuje się algorytm przydziału ciągłego.

Zamiast wykazu **n** wolnych adresów dyskowych można przechowywać:
-adres pierwszego wolnego bloku,
-liczbę **m** wolnych bloków następujących bezpośrednio po nim.

Każda pozycja na wykazie wolnych obszarów składa się z: -adresu dyskowego,
-licznika.

Każdy wpis zajmuje więcej miejsca niż zwykły adres dyskowy, lecz cały wykaz będzie krótszy, jeśli liczniki będą większe niż **1**.

10.6. Implementacja katalogu

Zarządzanie katalogiem wpływa na wydajność, działanie i niezawodność systemu plików.

❐ Lista liniowa

Katalog stanowi **listę nazw** plików **ze wskaźnikami** do bloków danych.

Znalezienie określonej pozycji na liście wymaga **przeszukiwania liniowego**.

Utworzenie nowego pliku rozpoczyna przeszukiwanie katalogu
(sprawdzanie czy w istniejących plikach nie ma takiej samej nazwy),
po czym dodaje się nowy wpis na końcu katalogu.

Usunięcie pliku wymaga ***odnalezienia jego nazwy*** i zwolnieniu przydzielonego obszaru.

Powtórne wykorzystanie pozycji w katalogu może nastąpić poprzez:

1. oznaczenie danej pozycji jako nieużywanej (nadając jej specjalną nazwę),
2. dodanie jej do wykazu wolnych pozycji katalogowych,
3. skopiowaniu ostatniej pozycji w katalogu na zwolnione miejsce.

Wiele SO realizuje ***programową pamięć podręczną*** do przechowywania ostatnio używanych informacji katalogowych, co minimalizuje dyskowe operacje.

Lista uporządkowana umożliwia przeszukiwanie binarne.

Trzymanie listy uporządkowanej wymaga przemieszczania sporych ilości informacji katalogowych.

Przydatną może okazać się struktura danych typu B-drzewo.

❐ Tablica haszowania

Wpisy katalogowe dla plików przechowywane są na **liście liniowej**, zaś ***wskaźniki*** *do* **wpisów** -dla **danego pliku** na liście liniowej- pobiera się z Tablicy ***haszowanej***.

index **i**	wpisy
...	...
11	**Plik-11**
...	.....
121	**Plik-121**
...	.....
....	**.....**

Indeksem Tablicy haszowania jest wartość obliczona na podstawie **nazwy pliku**.

Trzeba rozwiązać kolizje: -wyniki haszowania nazw dwóch plików indeksują to same miejsca.

Problemy implementacji **Tablicy haszowania**:
-stały rozmiar Tablicy,
-zależność funkcji haszowania od rozmiaru Tablicy

10.7. Efektywność, wydajność i spójność

Dysk jest główną przeszkodą szybkiego działania systemu, gdyż jest najwolniejszy.

❒ Efektywność

◗ W systemie UNIX **i-węzły** są wstępnie rozmieszczane na dysku.

Dysk traci część swojej przestrzeni.

Wstępny przydział i-węzłów i rozrzuceniu ich w obrębie strefy polepsza wydajność systemu.

∗W algorytmach Uniksowych **bloki danych** pliku występują blisko bloku z **i-węzłem** danego pliku, co oszczędza czas przeszukiwania dysku.

```
struct i-wezel {
   mode;            // typ pliku, katalog, specjalny..., 0- gdy wolny
   i_uid;           // identyfikator właściciela pliku
   i_size;          // rozmiar pliku w bajtach
   i_atime;         // ostatni czas dostępu
   i_ctime ;        // czas ostatniej zmiany i-wezla
   i_mtime;         // czas ostatniej zmiany pliku
   i_dtime;         // czas usunięcia pliku
   i_gid;           // identyfikator grupy właściciela pliku
   i_links_count;   // licznik otwartych dowiązań do pliku
   i_blocks;        // liczba  bloków danych pliku (po 512 B)
   i_flags;         // flagi pliku, m.in. prawa dostępu
   osdl             // informacje systemu operacyjnego
   i_block          // wskaźniki do bloków danych pliku
}
```

◗ Zapisuje się "**daty** Zapisu" aby określić, czy plik powinien być składowany, oraz "**daty** Odczytu", dzięki czemu wiadomo, kiedy korzystano z pliku.

Przechowywanie tych informacji wymaga zapisania pola w strukturze katalogowej przy każdym czytaniu pliku:

- przeczytanie bloku do pamięci,
- uaktualnienia jego fragmentu,
- zapisania bloku z powrotem na dysku.

◗ Wielkość wskaźników stosowanych przy dostępie do danych a efektywność systemu.

Część systemów korzysta się ze wskaźników o rozmiarze 16 lub 32 bity.

Takie rozmiary wskaźników ograniczają długość pliku do 2^{16} (64 KB) lub 2^{32} B (4 GB).

Można zaimplementować wskaźniki 64-bitowe, aby przesunąć to ograniczenie do 2^{64} B.

◗ **Dobór rozmiaru wskaźnika a skutki zmian technologicznych.**

MS-DOS był wyposażony w system plików, który zarządzał tylko obszarem 32 MB .
(wpis w tablicy FAT miał 12 bitów i wskazywał na grono o rozmiarze 8 KB).

Ze wzrostem pojemności dysków, dyskowe struktury danych i algorytmy w systemie MS-DOS uległy zmianie (wpisy w tablicy FAT rozszerzono do 16, a potem do 32 bitów).

Pierwotne decyzje dotyczące systemu plików były podyktowane względami efektywności.

W Solaris początkowo **T**ablica **P**rocesów oraz **T**ablica **O**twartych **P**lików miały długość ustaloną podczas rozruchu systemu.

Zapełnienie Tablicy **P**rocesów uniemożliwiało tworzenie dalszych procesów.

☛ Rozmiary Tablic mogły być zwiększane tylko przez powtórną kompilację jądra.

☛ Aktualnie Solaris ma prawie wszystkie struktury jądra przydzielane dynamicznie, przez co SO stał się wolniejszy.

❒ Wydajność

Sprzętowe sterowniki dyskowe zawierają pamięć ***cache*** mieszczącą całe ścieżki.

Odnalezioną ścieżkę czyta się do pamięci ***cache*** dysku.

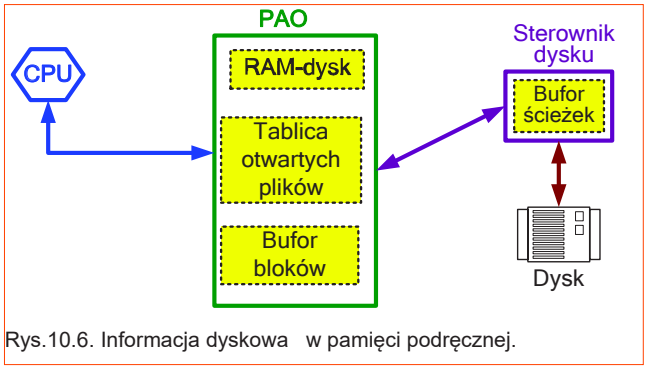
Sterownik dysku przesyła żądane sektory do SO.

System tworzy w **PAO** ***P**amięć **P**odręczną **B**loków **D**yskowych (disk cache),* w której przechowuje bloki **zakładając**, że wkrótce znów będą potrzebne.

Można całą ***nieużywaną*** PAO traktować jako pulę buforów, wspólną dla systemu stronicowania i systemu PAO- disk cache.

☛ SO wykonujący dużo operacji We/Wy będzie wykorzystywał większość buforów jako podręczna **pamięć bloków.**

☛ Niektóre systemy optymalizują podręczną pamięć bloków dyskowych, stosując różne algorytmy zastępowania bloków w zależności od rodzaju dostępu do pliku.



Rys.10.6. Informacja dyskowa w pamięci podręcznej.

➔ Bloki pliku o dostępie **sekwencyjnym** nie należy zastępować według algorytmu **LRU** *(Least Recently Used)*, ponieważ najnowsze bloki nie będą długo używane ➔

◗ Dostęp **sekwencyjny** można optymalizować za pomocą metod:

-wczesne zwalnianie: usuwanie bloku z bufora, gdy pojawi się zamówienie na następny blok.

Zakłada się, że poprzednie bloki nie będą ponownie używane - blokują miejsce w buforze.

-czytanie z wyprzedzeniem: przeczytanie wraz z zamawianym blokiem kilku następnych bloków i przechowanie ich w pamięci **cache**.

Przewiduje się, że będą one zamawiane po przetworzeniu bloku bieżącego.

W komputerach osobistych wydziela się część PAO jako ***dysk wirtualny*** - *RAM-dysk.*

Na RAM-dysku można wykonywać wszystkie operacje dyskowe.

Przechowuje się na nich pliki tymczasowe, jak pośrednie produkty pracy kompilatorów.

➔ Nadzór nad zawartością RAM-dysku sprawuje użytkownik.

☛ Podręczną pamięć bloków dyskowych kontroluje **SO**

❒ **Spójność**

Część informacji katalogowych przechowywana jest w PAO (cache).

Są to często informacje nowsze niż odpowiadające im informacje na dysku, gdyż zapisanie na dysku informacji katalogowych zgromadzonych w ***cache*** nie zawsze odbywa się wraz z ich uaktualnianiem.

✹ **Możliwe skutki awarii komputera.**

Tablica **O**twartych **P**lików z reguły zostaje utracona, a wraz z nią giną wszystkie zmiany wprowadzone w katalogach otwartych plików.

Stan niespójności systemu plików: bieżący stan niektórych plików staje się niezgodny z zapisami w strukturze katalogowej.

Podczas rozruchu systemu często wykonuje się specjalny program wykrywający i korygujący niespójności występujące na dysku.

Program sprawdzania spójności *(consistency checker)* porównuje dane w strukturze katalogowej z blokami danych na dysku i próbuje usunąć niezgodności.

Dla **listowego przydziału** jeżeli istnieje połączenie między każdymi dwoma blokami, to na postawie bloków danych można zrekonstruować cały plik i odtworzyć strukturę katalogową.

Dla **indeksowego przydziału** utrata wpisu katalogowego jest **tragiczna**, gdyż w blokach danych nie ma żadnych informacji dotyczących ich wzajemnego powiązania.

W systemie UNIX wpisy katalogowe używane do **Czytania** przechowywane są **podręcznie**.

Operacje **P**isania skutkujące **przydzieleniem** obszaru, powodują zapisanie bloku **i-węzła** ***przed*** zapisaniem bloków danych na dysku.

Dodanie do **SO tysięcy dodatkowych instrukcji aby zaoszczędzić kilku ruchów głowicy może okazać się korzystne.**

ANEKS 10.1. Pliki mapowane w pamięci w systemie Windows

Mapowanie pliku umożliwia rezerwację obszaru przestrzeni adresowej i przydzielenie do niej PAO.

☛ Po zmapowaniu pliku korzysta się z niego tak, jakby plik był załadowany do PAO.

◗ Zastosowania plików mapowanych w pamięci:

1. System używa je do ładowania i wykonywania plików .exe i DLL.
   Oszczędność miejsca w pliku wymiany i czasu potrzebnego na uruchomienie.
2. Mapowanie plików danych na dysku na przestrzeń adresową procesu
   Nie trzeba wykonywać operacji We/Wy ani buforować zawartości pliku.
3. Dzielenia danych przez wiele procesów działających na tym samym komputerze.
   Bardzo efektywny sposób komunikacji między procesami na jednej maszynie.

Dzielenie pamięci uzyskuje się w wyniku zmapowania przez dwa lub więcej procesów widoków **tego samego** **O**biektu**M**apowany**P**lik.

Wszystkie procesy **muszą** używać **tej samej Nazwy** **O**biektu**M**apowany**P**lik.

Efektem jest dzielenie przez procesy tych **samych ramek** pamięci fizycznej.

Jeśli jeden proces zapisuje dane w przestrzeni ***dzielonego*** **O**biektu**M**apowany**P**lik, wszelkie zmiany będą natychmiast widziane przez drugi proces w tej przestrzeni.

❒ **Pliki wykonywalne** i **DLL mapowane** w **PAO**

Gdy wątek wywołuje funkcję ***CreateProcess***, system wykonuje działania:

1. lokalizuje **plik .exe** podany w wywołaniu ***CreateProcess***.
2. tworzy w jądrze nowy **O**biekt**P**roces.
3. tworzy prywatną przestrzeń adresową dla nowego procesu.
4. rezerwuje obszar przestrzeni adresowej tak duży, aby pomieścić plik .exe.
   Pożądana lokalizacja tego obszaru jest podana w samym pliku .exe.
5. zapamiętuje, że pamięć fizyczna, na którą jest odwzorowany zarezerwowany obszar, należy do pliku **.exe** na dysku, a nie do systemowego pliku wymiany.
6. jeśli nie może zmapować pliku **.exe** ani wymaganych DLL, wyświetla odpowiedni komunikat i zwalnia przestrzeń adresową procesu oraz jego ObiektJądra.
   CreateProcess zwraca FALSE, funkcja *GetLastError* informacje o przyczynach błędu.

Po zmapowaniu **pliku.exe** system sięga do sekcji tego pliku, w której wymienione są biblioteki DLL z funkcjami wywoływanymi przez kod .exe i wywołuje *LoadLibrary* dla każdej z tych bibliotek:

a) rezerwuje obszar przestrzeni adresowej zdolny pomieścić plik DLL.
   Pożądana lokalizacja tego obszaru podana jest w pliku DLL.
   Standardowe biblioteki DLL Windows mają inne adresy bazowe.

b) jeśli nie może zarezerwować obszaru pod preferowanym przez DLL adresem bazowym system próbuje znaleźć inny obszar przestrzeni adresowej;

JEDNAKŻE SYSTEM:

-może w ogóle nie załadować tego pliku, jeśli nie ma w nim informacji o relokacji,

-musi dokonać przesunięć wewnątrz biblioteki DLL, co wydłuża czas załadowania.

➔ **Aby uruchomić** **APLIKACJĘ.exe** system wywołuje kolejno funkcje:

-**CreateFile**, aby otworzyć plik dyskowy ***.exe.**

-**CreateFileMapping**, aby utworzyć **O**biekt**M**apowany**P**lik.

-**MapViewOfFile*Ex*** (z flagą SEC_IMAGE) aby zmapować **plik.exe** na przestrzeń adresową nowego procesu.

Używa ****Ex***, gdyż obraz pliku ma być mapowany zgodnie z **adresem bazowym** podanym w obrazie **pliku.exe**.

System tworzy główny wątek procesu, umieszcza adres pierwszego bajta kodu wykonywalnego zmapowanego widoku we wskaźniku instrukcji wątku, rozpoczynając wykonanie aplikacji.

Po uruchomieniu **2-giej** instancji tej samej aplikacji, system stwierdza obecność **O**biektu**M**apowany**Plik** dla danego pliku **.exe** i **nie** tworzy nowego **O**biektu**P**liku ani **OMP**liku.

System po **raz drugi** mapuje widok pliku, ale tym razem w kontekście przestrzeni adresowej **nowo** utworzonego procesu.

W praktyce system mapuje identyczny plik na dwie przestrzenie adresowe jednocześnie.

W rezultacie procesy **dzielą te same strony pamięci fizycznej** zawierającej fragment wykonywanego kodu.

ANEKS 10.2. **Tablica przydziału plików** (*File Allocation Table* -**FAT**)

Jest to odmiana metody przydziału **listowego**.

Początkowa część **każdej strefy** na dysku zarezerwowana jest na przechowywanie **Tablicy**.

Tablica indeksowana jest [**numerami bloków**], i ma po **jednej** pozycji na każdy **blok** dyskowy.

Wpis katalogowy zawiera numer **pierwszego** bloku pliku.

Wpis katalogowy

Plik		Blok początkowy
File1	**...**	**123**

Pozycja w Tablicy **indeksowana** przez **numer danego** bloku, zawiera **numer następnego** bloku w pliku.

Łańcuch taki ciągnie się aż do ostatniego bloku, który ma na odpowiadającej mu pozycji w tablicy symbol końca pliku.

Na bloki nie używane wskazuje liczba **0** umieszczona na odpowiedniej pozycji w Tablicy.

Przydzielenie nowego bloku do pliku to:

-odnalezienie pierwszej **zerowej** pozycji w Tablicy,

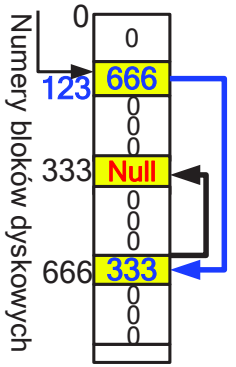
-zastąpienie poprzedniej wartości (koniec pliku) adresem nowego bloku.

Gdy tablicy FAT **nie ma w pamięci podręcznej**, to głowica dysku musi:

-przemieścić się na początek strefy, aby przeczytać dane w tablicy FAT i odnaleźć położenie potrzebnego bloku,

-przesunąć się do miejsca występowania tego bloku.

Na rysunku obok plik składa się z bloków o numerach: **123, 666 i 333**.



FAT polepsza czas dostępu swobodnego, gdyż głowica dyskowa może znaleźć położenie dowolnego bloku na podstawie informacji czytanych w Tablicy.