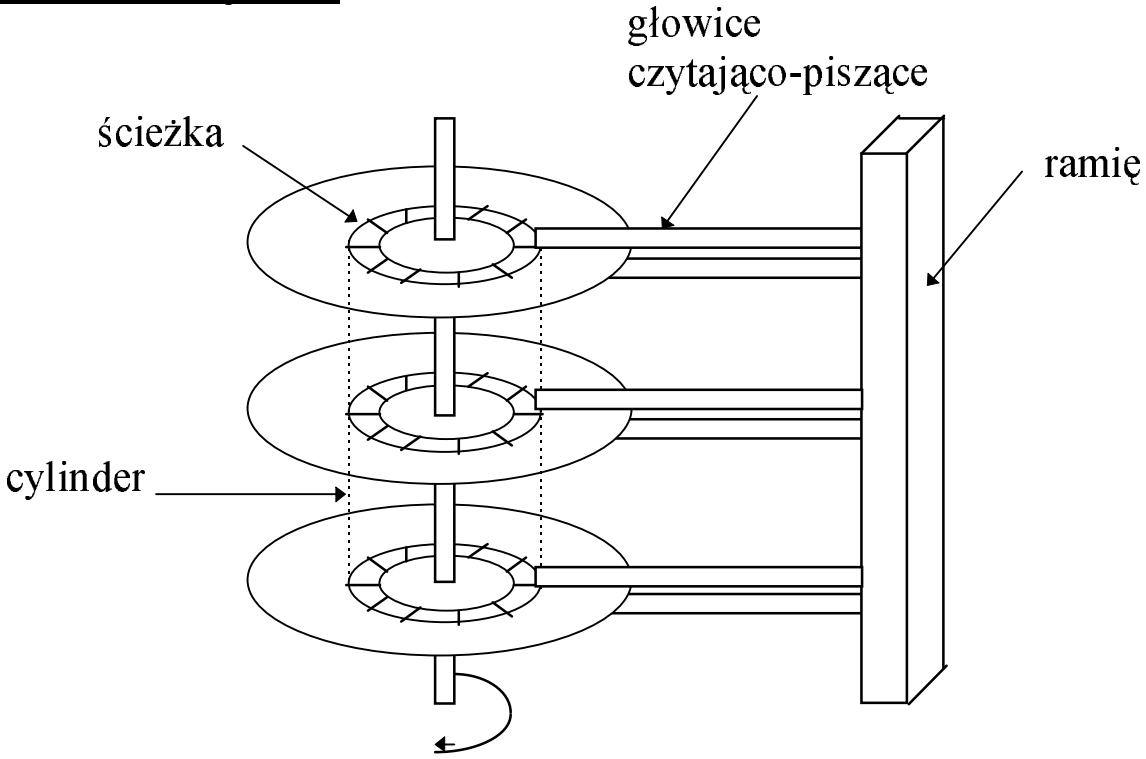
Zarządzanie pamięcią pomocniczą

Pamięć główna: zwykle za mała, ulotna
Pamięć pomocnicza: służy do trwałego przechowywania dużych ilości danych
Nośniki: taśma, dysk magnetyczny, dysk optyczny

Struktura dysku



Dysk o nieruchomych głowicach: osobna głowica dla każdej ścieżki (bęben: dysk z jednym cylindrem)
Dysk z ruchomą głowicą

- **Sektor**: najmniejszy blok, który można zapisać/odczytać z dysku
- **Ścieżka**: zbiór wszystkich sektorów na pojedynczej powierzchni leżących w tej samej odległości od osi obrotu dysku

- **Cylinder**: zbiór wszystkich ścieżek leżących w tej samej odległości od osi obrotu dysku na dysku w wieloma talerzami

Adres dyskowy: nr napędu, nr powierzchni, nr ścieżki (cylindra), nr sektora

- **Czas przeszukiwania** (ang. *seek time*): czas przesunięcia głowicy na właściwą ścieżkę (nast. elektronicznie wybiera się właściwą powierzchnię)
- **Czas oczekiwania** (ang. *latency time*): czas oczekiwania aż głowica znajdzie się nad właściwym sektorem (to dysk się kręci!); określony przez prędkość rotacji dysku
- **Czas transmisji** - czas potrzebny na odczytanie/zapisanie danych

Jednostka transmisji: sektor lub grupa sektorów

SO traktuje dysk jak jednowymiarową tablicę: nry bloków rosną wzdłuż sektorów na ścieżce, nast. wzdłuż ścieżek w cylindrze, nast. od cylindra 0 do ostatniego

s - liczba sektorów na ścieżce

t - liczba ścieżek w cylindrze

(powierzchnia j, cylinder i, sektor k) $\rightarrow$ blok b

$$b = k + s \cdot (j + i \cdot t)$$

Zarządzanie wolnymi obszarami dyskowymi

System przechowuje informacje o wolnych blokach dyskowych:

1. ***Mapa bitowa***

   Każdy blok jest reprezentowany przez jeden bit (0 $\Rightarrow$ blok wolny)

   - względnie szybko można odszukać n kolejnych bloków

- mało wydajne, gdy nie można przechowywać mapy w pamięci głównej

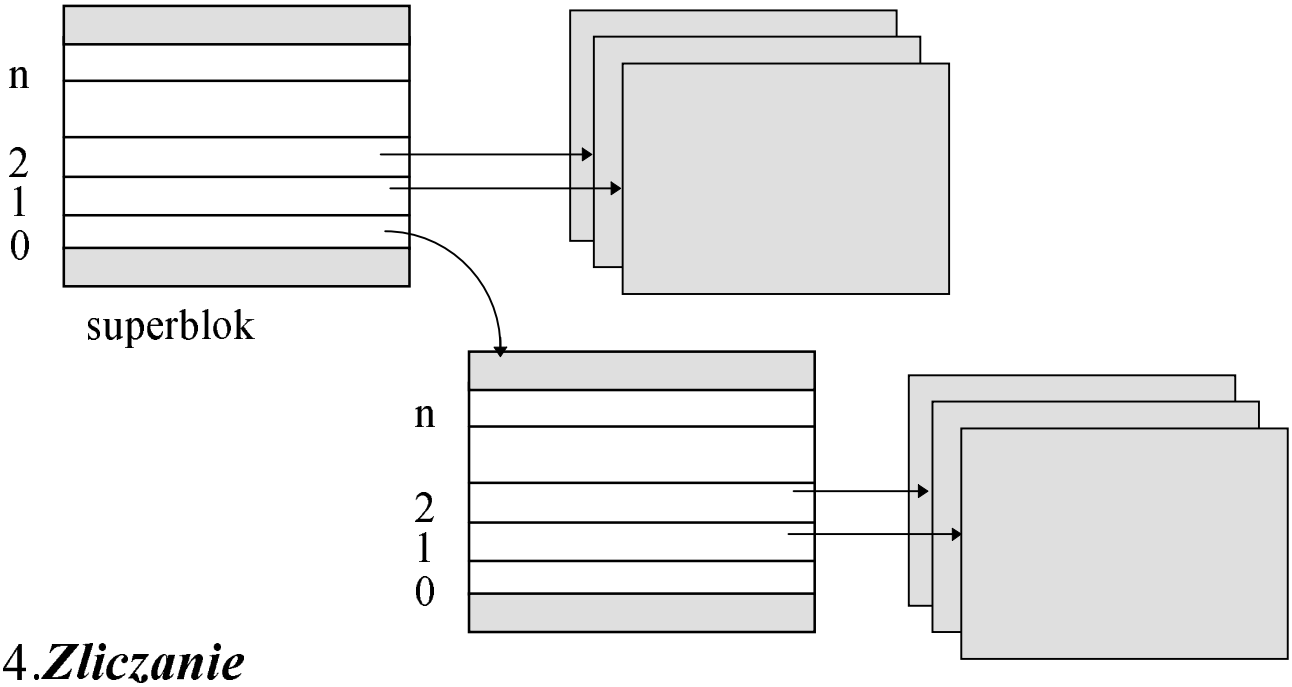
2. ***Lista powiązana***

   Każdy wolny blok zawiera wskaźnik do następnego wolnego bloku

   Mało wydajna - aby przejrzeć listę, trzeba odczytać każdy blok

3. ***Grupowanie***

   Przykład: Unix przechowuje w superbloku *n* adresów wolnych bloków; *n*-ty wolny blok zawiera kolejne *n* adresów itd.



4. ***Zliczanie***

   W każdym węźle listy pamięta się adres kolejnego ciągłego obszaru i liczbę bloków wchodzących w skład tego obszaru

Przydział miejsca na dysku

1. ***Przydział ciągły***

   Plik zajmuje ciągły obszar na dysku

   - minimalna liczba operacji dyskowych

- przydział określony za pomocą adresu początku i liczby bloków
- prosta implementacja dostępu sekwencyjnego i bezpośredniego
- trudno znaleźć miejsce na nowy plik (strategie: pierwszy pasujący, najlepszy pasujący; fragmentacja zewnętrzna)
- trudno rozszerzać plik (ew. trzeba z góry podać rozmiar pliku)
- kompresja

2. ***Przydział listowy***

Plik stanowi powiązaną listę bloków dyskowych

- przydział określony za pomocą adresu pierwszego i ostatniego bloku
- łatwe tworzenie i rozszerz. pliku, nie ma fragm. zewn.
- niepotrzebna kompresja ani deklarowanie rozmiaru pliku
- dostęp praktycznie tylko sekwencyjny
- dużo miejsca zajmują wskaźniki
- podatne na awarie

Odmiana przydziału listowego ***tablica przydziału plików*** (*FAT*), np. MS-DOS, OS/2

- po jednej pozycji dla każdego bloku dyskowego
- indeksowana numerami bloków
- przydział określony za pomocą numeru pierwszego bloku pliku
- pozycja w tablicy zawiera numer następnego bloku
- zero oznacza wolny blok

3. ***Przydział indeksowy***

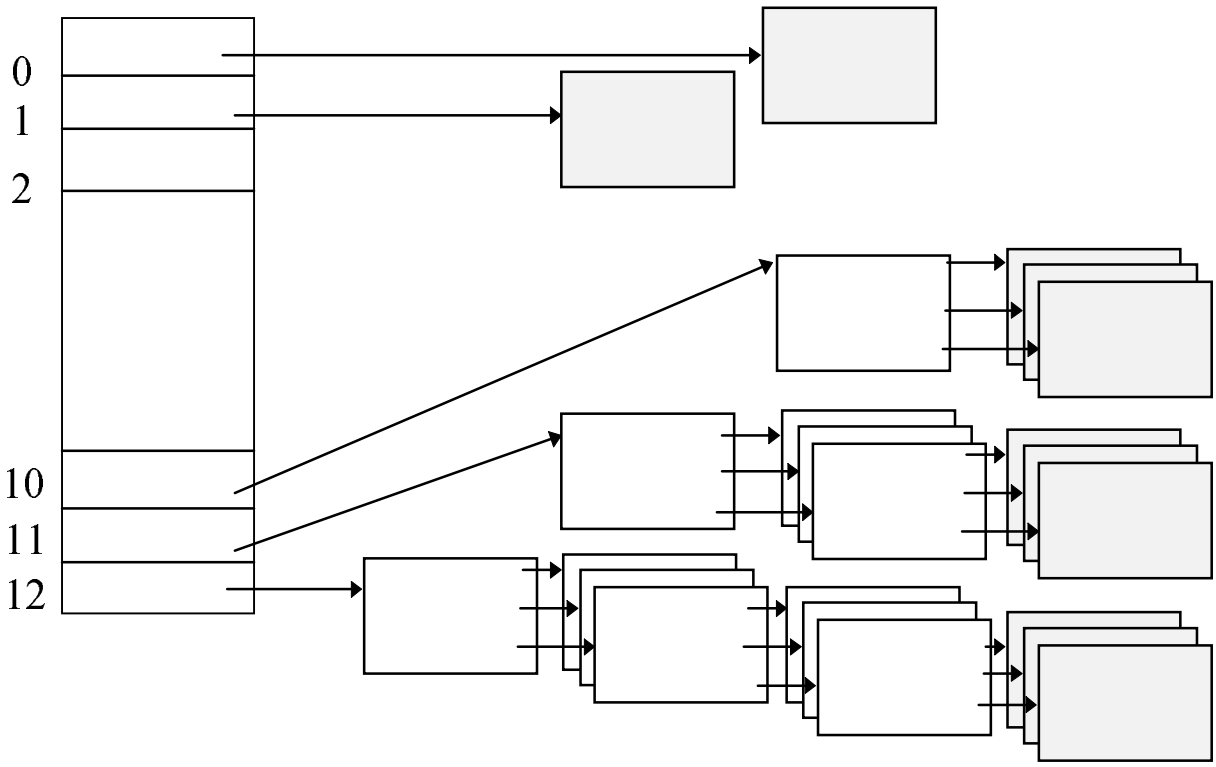
Każdy plik ma własny blok indeksowy będący tablicą adresów bloków dyskowych

- przydział określony za pomocą adresu bloku indeksowego
- pozycja o numerze *i* w tym bloku wskazuje *i*-ty blok pliku
- prosta implementacja dostępu sekwencyjnego i bezpośredniego
- nie ma fragmentacji zewnętrznej
- wskaźniki zajmują więcej miejsca niż przy przydziale listowym (zwykle część bloku indeksowego jest niewykorzystana)

Jak organizować bloki indeksowe?

- lista bloków indeksowych
- indeks wielopoziomowy
- schemat łączony

Przykład: Unix



Maksymalny rozmiar pliku:

- $10 \cdot 1KB + 256 \cdot 1KB + 256^2 \cdot 1KB + 256^3 \cdot 1KB \approx 16GB$
- w i-węźle 32 bity na rozmiar pliku $\Rightarrow$ 4GB

Szeregowanie żądań do dysku

czas obsługi żądania = czas przeszukiwania +
czas oczekiwania + czas transmisji

Charakterystyka żądania dyskowego:
- operacja wejścia czy wyjścia
- adres dyskowy
- adres w pamięci
- wielkość transmisji

W systemie wieloprogramowym w kolejce do dysku może oczekiwać więcej niż jedno żądanie

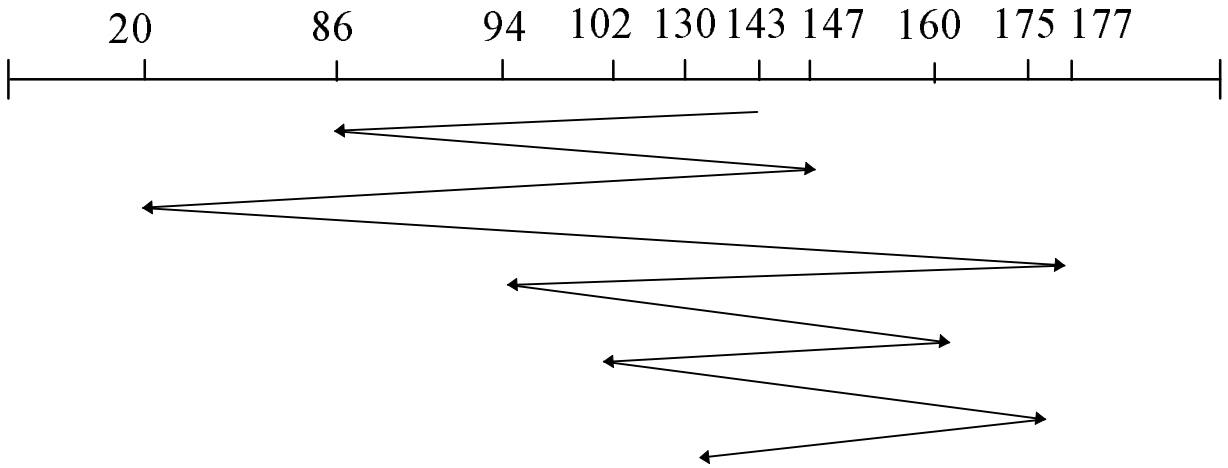
Przykład: ruchoma głowica dysku o 200 ścieżkach (0..199) obsługuje żądanie do ścieżki 143, a poprzednio zakończyła obsługę żądania do ścieżki 159. W kolejce czekają żądania (w kolejności przybycia):

86, 147, 20, 177, 94, 160, 102, 175, 130

Strategie szeregowania żądań dyskowych:

1. ***FCFS***

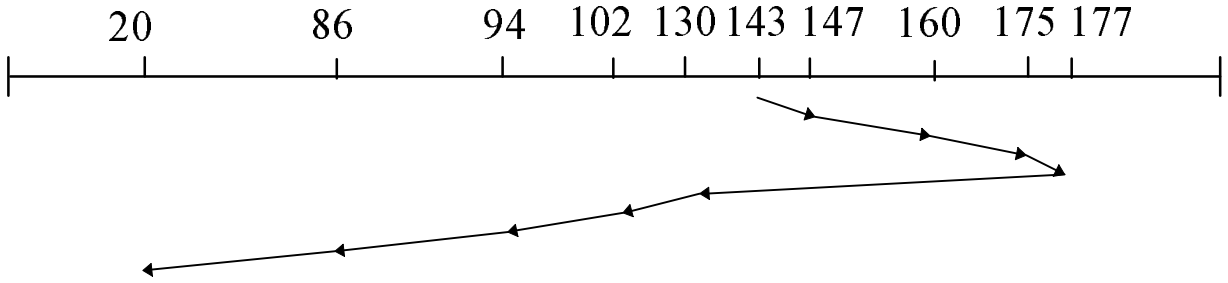
Obsługa w kolejności przybywania żądań



- prosta implementacja

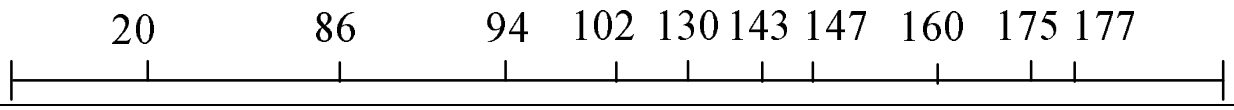
- sprawiedliwa (brak zagłodzenia)
- akceptowalna przy małym obciążeniu; przy dużym daje długi średni czas obsługi, choć małą wariancję

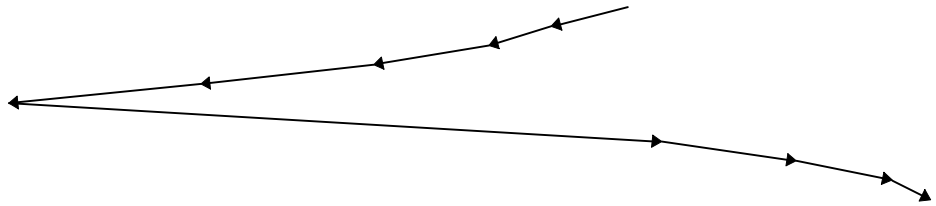
2. ***SSTF*** (ang. *Shortest Seek Time First*)
Do obsługi wybiera żądanie z najmniejszym czasem przeszukiwania względem bieżącej pozycji głowicy



- przepustowość lepsza niż przy FCFS; średni czas obsługi krótszy dla średniego obciążenia, ale wariancja większa
- nie jest optymalny
- możliwe zagłodzenie; skrajne ścieżki są dyskryminowane
- akceptowalny w sysemach wsadowych, ale nie akceptowalny w systemach interakcyjnych

3. ***SCAN*** („winda”)
Głowica startuje od jednego końca dysku i przemieszcza się w kierunku przeciwnego, obsługując żądania do mijanych ścieżek, aż dotrze do drugiego końca. Wówczas zmienia kierunek ruchu

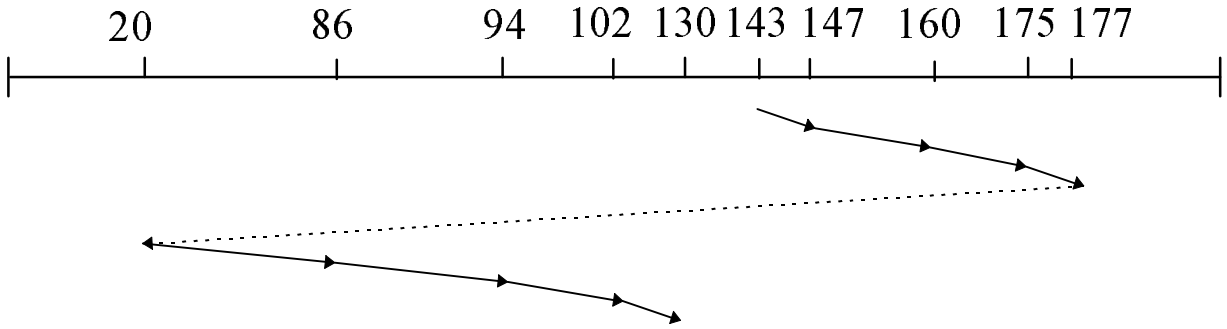




- mniejsza wariancja niż dla SSTF
- z chwilą gdy głowica zmienia kierunek, tuż przed głowicą będzie stosunkowo mało żądań do obsłużenia
- najczęściej spotykana w praktyce strategia
- dobra z punktu widzenia trzech miar (przepustowość, średni czas obsługi, wariancja czasu obsługi), choć skrajne ścieżki nadal nieco dyskryminowane

4. C-SCAN

Jak SCAN, ale żądania są obsługiwane tylko podczas ruchu głowicy w jednym kierunku (dysk traktowany jak powierzchnia cykliczna)



- mniejsza wariancja czasu oczekiwania
- żadne ścieżki nie są dyskryminowane
- badania symulacyjne wykazały, że najlepiej połączyć SCAN (przy małym obciążeniu) z C-SCAN (przy dużym obciążeniu)

5. *LOOK* i *C-LOOK*

Warianty (praktyczne) SCAN i C-SCAN - głowica przesuwa się do skrajnego żądania (nie ścieżki!) w każdym kierunku

6. ***F-SCAN*** (***N-Step SCAN***)
   Ruch głowicy jak w SCAN, ale są obsługiwane tylko te żądania, które nadeszły przed rozpoczęciem ruchu w danym kierunku
   - jeszcze mniejsza wariancja
   - nie ma zagłodzenia w sytuacji, gdy stale nadchodzą żądania do bieżącej ścieżki

7. Tworzenie kolejek do sektorów
   - ma sens jedynie w systemach w dużym obciążeniem
   - optymalizacja rotacyjna

Efektywność algorytmu szeregowania może zależeć od metody przydziału miejsca na plik, rozmieszczenia na dysku katalogów i bloków indeksowych (gdzie umieszczać?)