L12.Z1. Rozważamy następujące metody przydziału plików: **ciągłą**, **listową**, **indeksowaną**, **i-węzeł**. Dysponujesz **bitmapą wolnych bloków** i listą wszystkich i-węzłów. Czy i jak można naprawić poniższe błędy dla przydziału:

- ciągłego – uszkodzenie jednego z elementów pary,
- listowego – wskaźnik odnosi się do wolnego bloku lub zajętego bloku z innego pliku,
- indeksowanego – dwa indeksy w tablicy wskazują na ten sam blok,
- i-węzeł – uszkodzenie wskaźnika na blok pośredni.

bitmapa wolnych bloków - bitmapa która mówi czy blok jest wolny czy zajęty

ciągła alokacja – zapisanie każdego pliku w postaci ciągłego zbioru bloków dyskowych.

- zalety:
  - prosta implementacja – spamiętanie dwóch liczb: adresu dyskowego pierwszego bloku oraz liczby bloków w pliku
  - stały czas wyszukiwania, cały plik można wczytać z dysku w pojedynczej operacji (operacja `seek`)
- wady:
  - fragmentacja wewnętrzna, gdy plik nie zajmuje całego bloku
  - fragmentacja zewnętrzna, wymagane kompaktowanie; aby uzupełniać luki w blokach po usuwaniu plików, trzeba byłoby trzymać listę bloków (niby spoko, ale trzeba mieć pewność, ile miejsca zajmuje plik, żeby znaleźć blok odpowiedniego rozmiaru)
- i-węzeł przechowuje parę (pierwszy-blok, ostatni-blok)
- przydatne w przypadku nośników read-only, przykład: CD-ROM – rozmiar jest z góry znany i nigdy się nie zmienia
- płyty DVD: system plików wykorzystywany na płytach DVD – UDF (*Universal Disk Format*) wykorzystuje 30-bitową liczbę do zaprezentowania długości pliku, co ogranicza rozmiar plików do 1 GB. W konsekwencji filmy DVD zazwyczaj są zapisywane w postaci trzech lub czterech plików po 1 GB, z których każdy jest ciągły. Te fizyczne części pojedynczego pliku logicznego (filmu) są określane mianem obszarów (ang. *extent*).

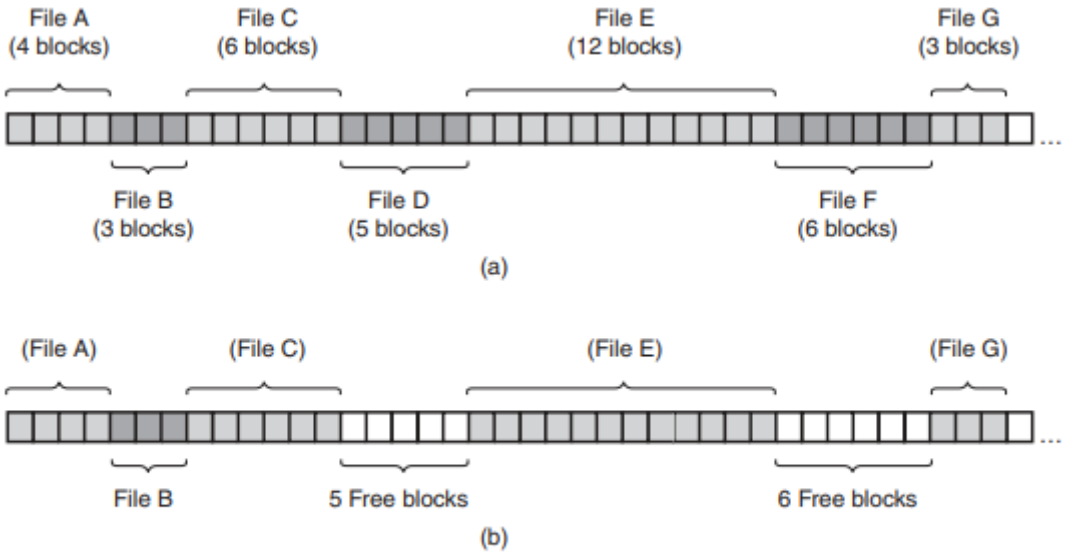

Figure 4-10. (a) Contiguous allocation of disk space for seven files. (b) The state of the disk after files *D* and *F* have been removed.

lista jednokierunkowa – przechowywanie każdego pliku w postaci jednokierunkowej listy bloków dyskowych.

- pierwsze słowo z każdego bloku jest używane jako wskaźnik do następnego bloku
- pozostała część bloku jest przeznaczona na dane
- zalety:
  - tylko fragmentacja wewnętrzna w ostatnim bloku
  - wystarczy, że wpis w katalogu będzie zawierał adres dyskowy tylko pierwszego bloku – resztę można znaleźć, począwszy od wskazanego miejsca -> łatwy odczyt sekwencyjny
- wady:
  - dostęp losowy bardzo wolny
  - ilość miejsca na dane w bloku nie jest potęgą 2, ponieważ kilka bajtów zajmuje wskaźnik – spadek wydajności, ponieważ wiele programów czyta i zapisuje dane w blokach, których rozmiar jest potęgą 2 – konieczny odczyt i konkatenacja informacji z dwóch bloków dyskowych

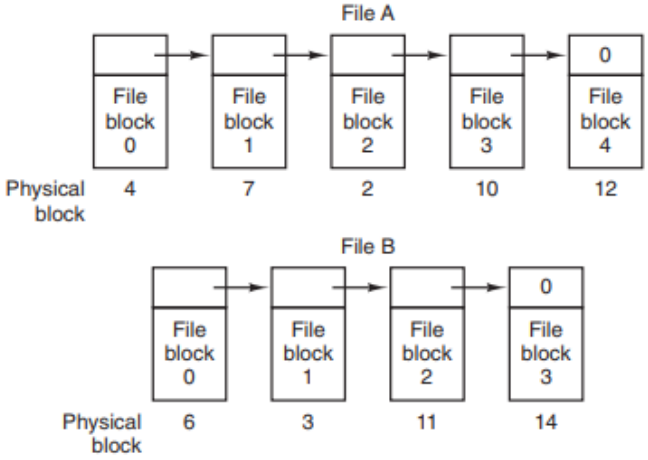

Figure 4-11. Storing a file as a linked list of disk blocks.

przydział indeksowany – odczytujemy słowo wskaźnika z każdego bloku dyskowego i umieszczamy go w tablicy w pamięci FAT (*File Allocation Table*)

- łańcuchy wykorzystania bloków dyskowych: A: 4, 7, 2, 10, 12, B: 6, 3, 11, 14 – kończą się specjalnym znacznikiem (np. -1), który nie jest prawidłowym numerem bloku
- zalety:
  - cały blok jest dostępny dla danych
  - losowy dostęp łatwiejszy
  - wystarczy, aby wpis w katalogu zawierał tylko numer początkowego bloku
- wady:
  - tabela musi cały czas być w pamięci – nie ma sensu dla dużych dysków

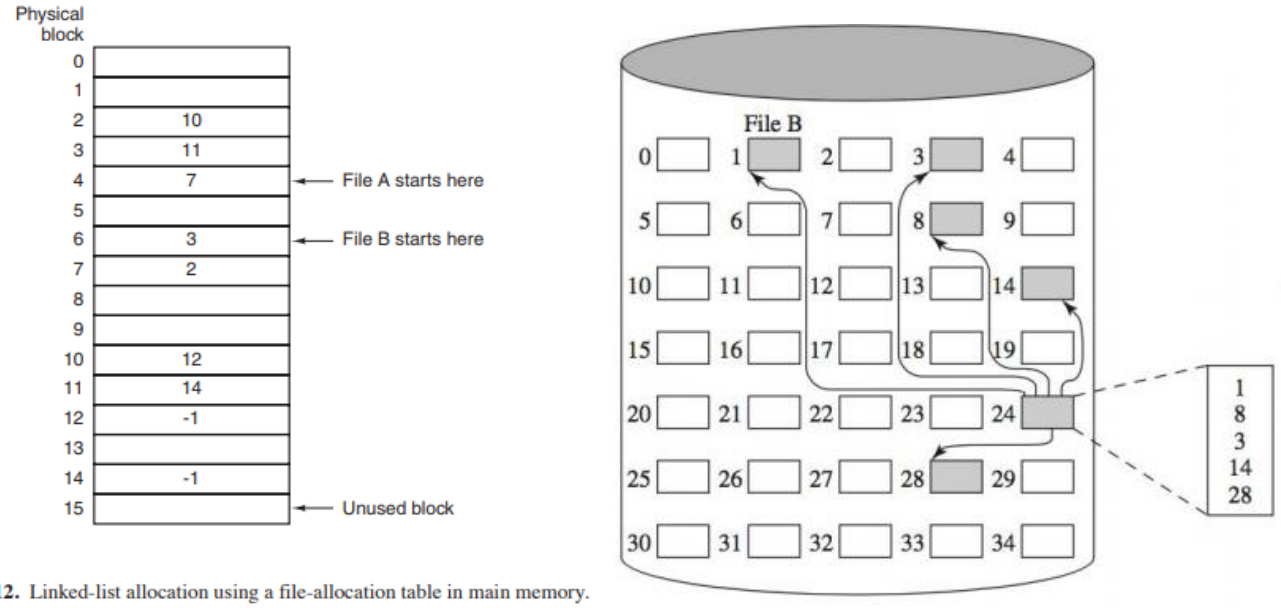

Figure 4-12. Linked-list allocation using a file-allocation table in main memory.

i-węzły – powiązanie z każdym plikiem i-węzła, w którym są zapisane atrybuty i adresy dyskowe bloków należących do pliku.

- zalety:
  - i-węzeł musi być w pamięci tylko wtedy, gdy powiązany z nim plik jest otwarty
  - dla odróżnienia od przydziału indeksowanego, gdzie tablica przeznaczona do zapisania jednokierunkowej listy wszystkich bloków jest proporcjonalna do rozmiaru samego dysku, mechanizm i-węzłów wymaga w pamięci tablicy o rozmiarze proporcjonalnym do maksymalnej liczby plików, które mogą być otwarte na raz – bez znaczenia, jaką pojemność ma dysk
- wady:
  - każdy z i-węzłów posiada miejsce na stałą liczbę adresów dyskowych – kłopot, kiedy plik rozrośnie się poza ten limit. Rozwiązanie: rezerwacja ostatniego adresu dyskowego nie na blok danych, ale na adres bloku zawierającego więcej adresów bloków dyskowych lub inne rozwiązanie: dwa bloki (lub więcej) zawierające adresy dyskowe lub nawet bloki dyskowe wskazujące na inne bloki dyskowe pełne adresów

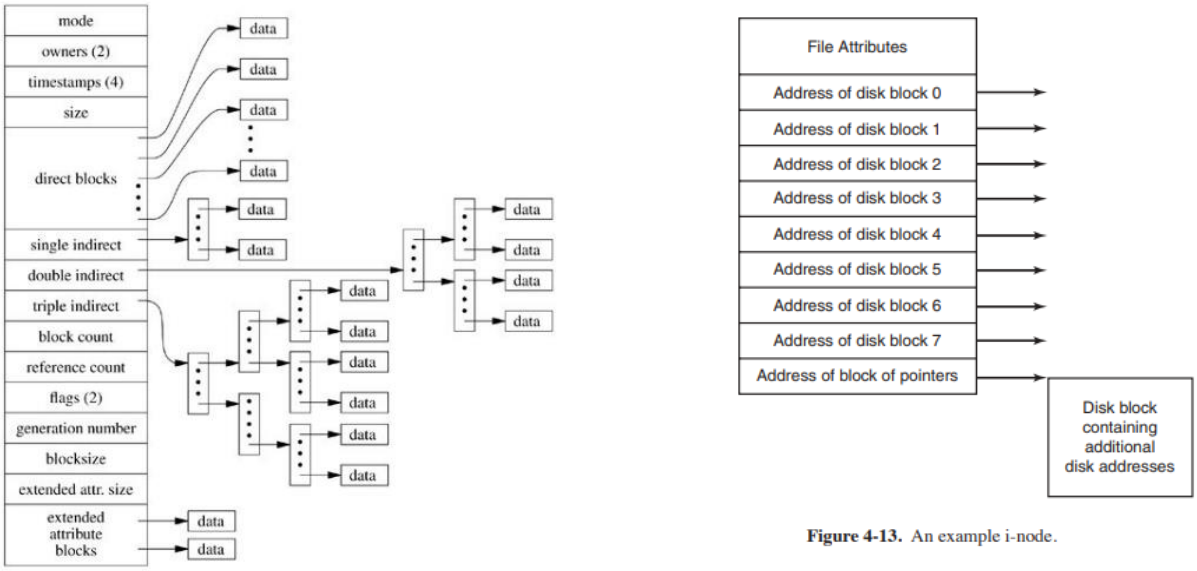

Figure 4-13. An example i-node.

Jak można naprawić

- ciągły – uszkodzenie jednego z elementów pary
  - adres dyskowy pierwszego bloku – szukamy poprzedniego w pamięci i-node'a, do jego adresu dodajemy liczbę jego bloków i otrzymujemy adres naszego inode'a
  - liczba bloków w pliku – szukamy następnego w pamięci inode'a, odejmujemy jego adres od adresu naszego – otrzymujemy liczbę bloków

- listowy – wskaźnik odnosi się do wolnego bloku lub zajętego bloku z innego pliku
  Można np. oprócz wskaźnika trzymać informację, do którego pliku dany blok należy – jeśli jestem w jakimś pliku i przechodzę do bloku, który jest w innym pliku, to widać, że dzieje się coś złego. Wtedy przechodzę po wszystkich blokach i sprawdzam, które bloki należą do pliku z moim identyfikatorem.

  Lub: mogę przejść po wszystkich wolnych blokach i innych zajętych inode (oprócz mojego) - to, czego nie przejdę (żaden blok nie ma na niego wskaźnika), to mój zepsuty plik.

- indeksowany – dwa indeksy w tablicy wskazują na ten sam blok

  Znowu zapisywać, do którego pliku należy, ale tu problem, bo nie wiemy, który blok przepisać.

  Lub: Ponownie przechodzę po wszystkich wolnych i zajętych (ale nie po naszym pliku), to co zostanie to nasz blok (ale gdzie go wstawić?). Lub: mogę wyrzucić jeden z tych indeksów.

- i-węzeł – uszkodzenie wskaźnika na blok pośredni

  Każdy blok może pamiętać swojego rodzica w drzewie. Można przejść po wszystkich blokach i sprawdzić, kto ma za rodzica ten blok (kosztowne, ale jedyne sensowne rozwiązanie).

L12.Z3. Rozważamy **liniową organizację katalogów**. Pokaż jak wyszukiwać, dodawać, usuwać i zmieniać nazwę **wpisów katalogów**. Kiedy warto przeprowadzić operację **kompaktowania** katalogu? W jakich przypadkach możliwe jest odwrócenie operacji skasowania pliku (ang. *undelete*)?

liniowa organizacja katalogów – katalog jest zorganizowany jako lista wpisów o plikach (rekordów)

Pokaż jak wyszukiwać, dodawać, usuwać i zmieniać nazwę wpisów katalogów.

wpis katalogu - zawieraja informacje takie jak: długość wpisu, numer i-node, nazwę pliku, długość nazwy pliku, typ pliku, (ewentualnie jakiś padding do wyrównania)

wyszukiwanie:
chcemy dopasować nazwę pliku do numeru i-node. Przechodzimy po liście i sprawdzamy.

dodawanie:
trzeba sprawdzić, czy jest miejsce w katalogu, by dodać wpis pliku. Jeśli nie ma, trzeba kompaktować i wtedy spróbować dodać wpis. Jeśli nadal nie ma miejsca, to możemy zrobić jakiś indirect wpis.

usuwanie:
ustawiamy pole długości poprzedniego rekordu na taką, aby „pomijała" usuwany plik

zmiana nazwy:
zmiana pola w rekordzie. Jeśli nazwa jest za długa, to może trzeba utworzyć nowy rekord. Jeśli dalej się nie mieści, trzeba to zapisać w innym bloku.

Kiedy warto przeprowadzić operację kompaktowania katalogu?

kompaktowanie – przesuwanie zajętych bloków tak, żeby możliwie jak najwięcej wolnych bloków połączyło się w ciągłe obszary, wymaga kopiowania bloków za wolnymi lukami

Kompaktowanie warto przeprowadzić, próba dodania nowego wpisu w sektorze kończy się niepowodzeniem z powodu zbyt dużej fragmentacji zewnętrznej (być może po kompaktowaniu się uda).

W jakich przypadkach możliwe jest odwrócenie operacji skasowania pliku?

Undelete można zrobić, gdy miejsce w którym był wpis o danym pliku w katalogu nie zostało nadpisane nowymi danymi.

Dane pliku w sektorach dysku również nie mogą być nadpisane. Czyli musi istnieć jego i-node, który nie został nadpisany.

Jeśli były co najmniej 2 hardlinki, to tak będzie, bo licznik referencji i-node'a nie spadnie do 0 a do 1. Kiedy licznik i-node spada do 0, to jest zwalniany.

Na windowsie usunięte pliki przenoszone do osobnego katalogu (kosza).