UMCS

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
W LUBLINIE
Wydział Matematyki, Fizyki i Informatyki

Kierunek: **Informatyka**

Bartłomiej Poleszak
nr albumu: 250606

System plików dla pamięci USB

Filesystem for USB flash drives

Praca licencjacka
napisana w Zakładzie Układów Złożonych i Neurodynamiki
pod kierunkiem dra Jacka Krzaczkowskiego

Lublin rok 2015

Spis treści

Wstęp

System plików jest częścią systemu operacyjnego, za pośrednictwem której użytkownicy i programy mogą dostać się do zasobów zgromadzonych w pamięci trwałej (takiej jak m. in. dyski twarde, dyski SSD, płyty DVD czy pamięci USB) komputera.

Chcąc dokładnie przeanalizować działanie konkretnego systemu plików początkujący programista może napotkać szereg problemów. Okazuje się, że te, które mają prostą budowę, były pisane w nierozwijanych aktualnie językach programowania. Konieczność poznania nowego języka może zniechęcić do zapoznania się z kodem źródłowym. Współczesne systemy plików są najczęściej pisane w języku C. Jest on bardzo popularny i nadal rozwijany, jednak systemy wykorzystujące go są dużo bardziej skomplikowane od starszych odpowiedników.

Celem tej pracy było stworzenie systemu plików, którego działanie będzie łatwe do analizy, a do jego implementacji posłuży współczesny język programowania - C++. W efekcie powstał system plików BPFS.

Poszczególne nośniki danych posiadają charakterystyczne dla nich właściwości fizyczne, a także mogą się różnić sposobem wykorzystania (mogą być pamięcią wewnętrzną lub pamięcią przenośną). Wszystko to powinno zostać uwzględnione podczas tworzenia nowego systemu plików. Dlatego też BPFS był projektowany z myślą o konkretnym nośniku - pamięciach USB.

W pierwszym rozdziale tej pracy przedstawiono czym są systemy plików i jakie zadania są przed nimi stawiane, a także wyszczególniono istotne cechy niektórych popularnych nośników danych. Dalsza część rozdziału jest przeglądem przykładowych systemów plików - zarówno historycznych, jak i używanych współcześnie.

Drugi rozdział jest szczegółowym opisem BPFS. Przedstawiono cele, do których miał dążyć ten systemem plików i ich realizację po stronie projektu. Zaprezentowano sposób przechowywania danych na pojedynczym nośniku i na macierzy składającej się z wielu pamięci USB.

Implementacja BPFS jest tematem trzeciego rozdziału. Oprócz opisu i instrukcji użycia aplikacji koniecznych do skorzystania z BPFS wymienione zostały narzędzia, które wspomagały proces ich tworzenia. Szczególną uwagę poświęcono bibliotece FUSE.

Ostatni, czwarty rozdział został poświęcony wizji rozwoju BPFS. Wskazane zostały funkcjonalności, które mogą poszerzyć możliwości przedstawionego systemu plików. Zaproponowano także narzędzia, dzięki którym BPFS może zostać przekonwertowany na inne systemy operacyjne.

Rozdział 1

Wprowadzenie do systemów plików

1.1 Czym jest system plików?

Wszędzie tam, gdzie istnieje potrzeba przechowywania dużej ilości danych pojawia się problem zorganizowania ich w taki sposób, by były jak najłatwiej dostępne i użyteczne.

Przykładem mogą być biblioteki - choć aktualnie wykorzystuje się komputery jeszcze do niedawna nie było niczym nadzwyczajnym, że aby znaleźć interesującą nas książkę musimy przeszukać szufladki pełne kart posegregowanych alfabetycznie lub tematycznie.

Podobny problem powstał kiedy nośniki danych takie jak dyskietki czy taśmy magnetyczne zaczęły osiągać coraz większe pojemności. Systemy operacyjne potrzebowały usystematyzowanego sposobu dostępu do pamięci trwałej i właśnie tą rolę spełnia system plików.

1.1.1 Zadania stawiane przed systemem plików

System plików powinien udostępniać zawartość nośnika danych pod postacią **plików**. Plikiem nazywamy ciąg bitów będący logiczną jednostką informacji. Często zachodzi potrzeba widocznego oddzielenia jednego zbioru plików od innego, choćby ze względu na podział tematyczny posiadanych informacji. W tym celu większość systemów plików pozwala na tworzenie **struktury katalogów**. Katalog jest listą **wpisów** (rekordów) wskazujących na inne pliki (także inne katalogi). W opisywanych w dalszej części pracy systemach plików katalog jest szczególnym rodzajem pliku.

Każdy plik ma swoją nazwę, za pomocą której użytkownik ma możliwość odwołania się do niego. Poza tym może posiadać szereg atrybutów opisujących jego stan bądź znaczenie dla systemu, np. czas utworzenia, czas ostatniej modyfikacji, rozmiar, położenie na dysku.

Systemy operacyjne z reguły pozwalają na korzystanie z nich za pośrednictwem kont użytkowników. Aby zapobiec sytuacjom gdy jeden użytkownik niszczy (celowo bądź nie) pracę drugiego implementowane są mechanizmy praw dostępu, w których każdy plik posiada zbiór atrybutów opisujący co dany użytkownik może z nim zrobić.

Istotnym zadaniem systemu plików jest przydzielanie przestrzeni dla nowych danych. Podczas normalnego użytkowania komputera z dużą częstotliwością powstawać będą nowe pliki, a wiele starych lub tymczasowych będzie usuwanych. System plików powinien zapewnić możliwość korzystania z przestrzeni zwolnionej przez usunięte pliki i robić to w wydajny sposób.

1.1.2 Cechy wynikające z budowy nośników danych

We współczesnych komputerach system operacyjny otrzymuje dostęp do nośników danych pod postacią jednolitego ciągu bitów, jednak aby osiągnąć jak najlepszą wydajność dostępu do informacji niezastąpiona okazuje się wiedza o fizycznych cechach urządzeń przechowujących dane.

W komputerach osobistych wciąż najpopularniejszym magazynem danych są magnetyczne **dyski twarde** (HDD - hard disk drive). Przechowują one dane na talerzach, które podczas pracy obracają się ze stałą prędkością, a zapis i odczyt jest dokonywany za pośrednictwem głowic umieszczonych na ruchomym ramieniu. Dane zorganizowane są w cylindry, które zawierają po jednej ścieżce dla każdej z głowic. Ścieżki podzielone są na sektory.

Ze względu na taką budowę nośnika najszybciej będą odczytywane dane znajdujące się w jednym cylindrze, gdyż jego zmiana wiąże się z czasochłonnym przesunięciem głowicy - dlatego system plików powinien w miarę możliwości zapisywać dane w ciągłej przestrzeni.

Inną cechą istotną przy projektowaniu systemu plików jest rozmiar sektora. Typowy sektor mieści 512 bajtów i jest odczytywany w całości nawet jeśli istotna dla nas informacja to ułamek jego rozmiaru. Mając to na uwadze systemy plików organizują udostępnioną im przestrzeń w **bloki**, będące najmniejszą możliwą ciągłą porcją danych. Rozmiar bloku jest wielokrotnością rozmiaru sektora dysku.

Coraz większą popularność zyskują półprzewodnikowe dyski SSD (solid state drive). Są one kilkukrotnie szybsze od tradycyjnych dysków twardych między innymi dzięki temu, że nie posiadają żadnych części ruchomych - składają się z kości pamięci podobnych do stosowanych w kluczach USB czy kartach pamięci. Cechuje je jednak inna wada - komórki pamięci wytrzymują ograniczoną liczbę zapisów. System plików uwzględniający tą cechę powinien promować zapisywanie w najdłużej nieużywanych rejonach pamięci.

Tabela 1.1: Wpis w katalogu systemu plików CP/M

bajty	**rozmiar**	**zawartość**
0	1	identyfikator użytkownika
1 - 8	8	nazwa pliku
9 - 11	3	rozszerzenie
12	1	numer wpisu (ekstentu)
13 - 14	2	nieużywane
15	1	liczba zajmowanych bloków
16 - 31	16	adresy bloków

1.2 Historia systemów plików

1.2.1 System plików CP/M

Gary Kildall stworzył system operacyjny CP/M dla procesora Intel 8080 w 1973 roku, a jego późniejsze wersje były projektowane dla procesora Z80. System plików CP/M charakteryzuje się przede wszystkim prostotą i szybkością działania. Opis jego budowy można odnaleźć w [1].

W systemie plików CP/M dla każdego dysku istnieje jeden katalog o zależnej od implementacji, ale ograniczonej ilości wpisów. Każdy rekord ma rozmiar 32B, a jego dokładna struktura została przedstawiona w tabeli 1.1. System operacyjny wspiera obsługę wielu kont, stąd każdy plik ma przypisany identyfikator użytkownika. Dzięki temu system może ukryć przed aktualnie zalogowanym pliki pozostałych, pomimo tego że wszystkie znajdują się w jednym katalogu. System obsługuje ośmioznakowe nazwy plików wraz z trójznakowym rozszerzeniem.

Struktura rekordu przewiduje miejsce dla szesnastu adresów jednokilobajtowych bloków. Limit 16kB do wielu zastosowań był niewystarczający nawet w czasach świetności CP/M, dlatego każde kolejne 16kB mogło zostać przydzielone dzięki wykorzystaniu kolejnego wpisu w katalogu.

Nigdzie na nośniku nie jest przechowywana informacja o wolnych blokach danych, zamiast tego CP/M odczytuje wszystkie wpisy z katalogu i zaznacza na mapie bitowej (w pamięci operacyjnej) zajęte bloki. Rozmiar pliku określany jest z dokładnością do kilobajta, przez co na poziomie systemu nie da się określić jego dokładnego rozmiaru. Koniec pliku musi być rozpoznany na podstawie jego struktury, ale ta nie jest narzucana przez system plików.

1.2.2 System plików zaprojektowany dla Multics

Projekt systemu operacyjnego Multics został zaprezentowany w 1965 roku. Jedną z innowacji jakie wprowadził była hierarchiczna struktura zastosowa-

nego w nim systemu plików. Jego opis został przedstawiony w [2], a dużo szczegółowych informacji na jego temat można odnaleźć w [3].

Plik jest w nim definiowany jako sekwencja elementów, gdzie elementem mogą być słowa maszynowe, znaki lub pojedyncze bity. Posiada nazwę, dzięki czemu użytkownik może komunikować się z systemem w celu tworzenia nowych plików, odczytu, modyfikacji bądź ich usuwania. Zapewniony jest swobodny dostęp, czyli do konkretnego fragmentu pliku można się dostać znając jego nazwę i porządkowy numer elementu. Projektanci zaznaczyli także, że format pliku powinien byc interpretowany przez moduły wyższego poziomu lub programy uruchamiane przez użytkownika, a sam system plików traktuje wszystkie w ten sam sposób.

Katalog jest specjalnym plikiem, który zawiera listę wpisów. Wpis może przybrać dwojaką postać:

- **gałąź** (branch) - wskazuje bezpośrednio na miejsce, gdzie znajduje się plik, a także charakterystyczny dla niego opis
- **link** - wskazuje wpis w tym samym lub innym katalogu

Linki mogą wskazywać na inne linki, co może doprowadzić do powstania pętli.

Opis w gałęzi zawiera czasy ostatniej modyfikacji i dostępu, informacje o ilości użytkowników korzystających z pliku w danym momencie, o trybie w jakim jest otwarty (odczyt/zapis) i o prawach dostępu dla poszczególnych użytkowników. W przypadku linków cały opis jest dziedziczony po odpowiadającej mu gałęzi.

Każda gałąź zawiera listę kontroli dostępu (access control list), w której zapisani są wszyscy użytkownicy i grupy wraz z pięcioma atrybutami dostępu (TRAP, READ, EXECUTE, WRITE, APPEND). Wszystkie z nich mogą być włączone lub wyłączone.

Kiedy użytkownik chce uzyskać dostęp do pliku i flaga TRAP jest dla niego aktywna wywoływana jest procedura z odpowiedniej listy pułapek. Pułapki mogą być zakładane przez użytkowników w celu dodatkowego zabezpieczenia. Przykładem może być komenda `LOCK`, która wymusza podanie hasła przy odwołaniu się do gałęzi.

Dla pozostałych atrybutów należy wziąć pod uwagę specjalny rodzaj pliku jakim jest katalog. Włączony READ pozwala na odczytanie pliku (w przypadku katalogu - wylistowanie zawartości), WRITE umożliwia modyfikację zawartości bez dopisywania na końcu (modyfikacja istniejącego wpisu w katalogu), APPEND udostępnia możliwość dopisywania nowych treści bez zmian w istniejącej (dodawanie nowych wpisów do katalogów). Kiedy plik ma włączoną flagę EXECUTE można go uruchomić, czyli system operacyjny podejmie próbę zinterpretowania go jako program. W przypadku katalogu pozwala na przeszukanie zawartości.

Tabela 1.2: Długości adresu w różnych rodzajach FAT

System plików	długość adresu
FAT12	12 bitów
FAT16	16 bitów
FAT32	28 bitów
exFAT	64 bity

Plik w Multiksie może osiągnąć rozmiar jednego megabajta, jednak istnieje sposób, by traktować katalog zawierający kilka plików jako jeden plik logiczny.

1.2.3 FAT - File Allocation Table

FAT występuję w kilku wariantach różniących się głównie długością adresu klastra (patrz - tabela 1.2). System (w wersji **FAT16**) zyskał popularność dzięki MS-DOS i był używany także we wczesnych wersjach systemu Windows (od Windows 95 także **FAT32**). **FAT12** jest standardem dla dyskietek 1.44MB i 2.88MB. FAT32 do tej pory jest używany w wielu urządzeniach przenośnych i obsługiwany przez wszystkie popularne systemy operacyjne.

Aktualnie Microsoft promuje odmianę nazwaną **exFAT**, jako rozwiązanie dla przenośnych pamięci flash ze względu na mniejszą ilość metadanych w porównaniu do NTFS. Oprócz tego wykorzystanie 64 bitowego adresowania pozwoliło pokonać czterogigabajtowe ograniczenie wielkości pliku z FAT32.

Opis sposobu działania FAT można odnaleźć w [4], [5], [6] a także [7].

Partycję FAT możemy podzielić na:

- **Sektor rozruchowy** - pierwszy fizyczny sektor (512B) partycji, jest odczytywany przy próbie rozruchu systemu operacyjnego
- **Tablicę FAT** w dwóch kopiach na wypadek uszkodzenia systemu plików
- **Katalog główny** - w FAT12 i FAT16 katalog główny znajduje się zaraz za tablicami alokacji plików, w FAT32 jest traktowany jak każdy inny katalog, a jego adres jest zapisany w sektorze rozruchowym.
- **Dane** podzielone na klastry - tutaj zapisywane są wszystkie pliki i pozostałe katalogi.

Głównym elementem tego systemu jest zawarta w nazwie tablica alokacji plików. Każdy jej element odpowiada jednemu klastrowi i wskazuje na następny klaster danego pliku. Tabela 1.3 przedstawia przykładową tablicę

Tabela 1.3: Przykładowa tablica alokacji plików przy szesnastobitowej adresacji

element	0000	0001	0002	0003	0004	0005	0006
wartość	FFFF	FFFF	0003	0004	0005	0009	0007

element	0007	0008	0009	000A	000B	000C	...
wartość	0008	FFFF	000A	000B	000C	FFFF	...

Tabela 1.4: Wpis w katalogu FAT

bajty	**rozmiar**	**Zawartość**
0 - 7	8	nazwa pliku
8 - 10	3	rozszerzenie
11	1	atrybuty
12 - 21	10	zarezerwowane dla systemu operacyjnego
22 - 23	2	czas modyfikacji
24 - 25	2	data modyfikacji
26 - 27	2	numer pierwszego klastra pliku
28	4	rozmiar w bajtach

FAT. Zakładając że plik rozpoczyna się w szóstym klastrze możemy odczytać w odpowiedniej komórce tablicy, że dalsza część pliku znajduje się w siódmym, a kolejna w ósmym. Odpowiadająca ósmemu klastrowi komórka ma wartość FFFF, co oznacza że jest to ostatni fragment tego pliku. Kiedy rozważymy plik zaczynający się na drugim klastrze i w podobny sposób śledząc fragmenty pliku odczytujemy że zajmuje on bloki 2, 3, 4, 5, 9, A, B, C. Jak widać na tym przykładzie zastosowana struktura pozwala na to, by pojedynczy plik nie musiał zajmować ciągłego obszaru nośnika - może być w dowolny sposób pofragmentowany, co jednak negatywnie wpływa na wydajność odczytu. W zamian za to ułatwione jest powiększanie plików.

Wpis w katalogu ma stałą długość - 32 bajty. Początek wpisu to nazwa pliku w formacie MS-DOS (8B + 3B rozszerzenia). Kolejny bajt zawiera flagi atrybutów, od których zależy czy plik jest tylko do odczytu, ukryty, systemowy i czy jest przeznaczony do archiwizacji. Plik systemowy jest również ukryty, ale dodatkowo zabezpieczony przed usuwaniem. Flaga archiwizacji nie jest używana przez system operacyjny - z założenia programy edytujące pliki powinny ją ustawiać, a archiwizujące wyłączać. Możliwym zastosowaniem jest unikanie kopiowana niezmienionych plików przy tworzeniu przyrostowych kopii zapasowych. Po atrybutach występuje dziesięciobajtowa pusta przestrzeń (może być wykorzystana przez system operacyjny), a następnie

czas i data ostatniej modyfikacji (bądź utworzenia) pliku. Dwa ostatnie pola to numer pierwszego klastra danych i rozmiar pliku w bajtach.

1.2.4 ISO 9660 (CDFS)

Płyty kompaktowe były tworzone głównie z myślą o dystrybuowaniu muzyki, jednak pojemność nośnika wzbudzała nadzieję na możliwość wykorzystania go także do przechowywania innych dużych porcji danych. W celu unormowania sposobu przechowywania danych komputerowych na płytach CD powstał standard zaproponowany przez Ecma International [8], a zaakceptowany przez Międzynarodową Organizację Normalizacyjną w 1988 roku - ISO 9660. W [4] można znaleźć opis tego systemu plików wraz z popularnymi rozszerzeniami.

Płyty kompaktowe są przenośnym nośnikiem, przez co ISO 9660 był tworzony z myślą o obsłudze jak największej ilości systemów operacyjnych. W związku z tym powstały trzy poziomy (*levels of interchange*) standardu:

- **Poziom 1** zapewnia kompatybilność z MS-DOS narzucając ograniczenia na długość nazw plików (8 znaków nazwy i 3 znaki rozszerzenia, w przypadku katalogów 8 znaków bez rozszerzenia). Poza tym wymusza, aby pliki zajmowały ciągłą przestrzeń na nośniku.
- **Poziom 2** znosi ograniczenia co do nazw plików.
- W **Poziomie 3** nie obowiązują żadne z powyższych ograniczeń.

W odróżnieniu od dysków twardych płyty CD nie są fizycznie podzielone na sektory, stosowany jest jednak logiczny podział na 2352 bajtowe bloki, z czego 2048 bajtów jest przeznaczone na dane. ISO 9660 pozwala na dzielenie płyty na partycje oraz obsługuje zbiory płyt (jeden system plików na wielu nośnikach).

Pierwsze 16 bloków płyty nie jest wykorzystywane przez system plików (standard opisuje je jako przestrzeń systemową). Od 17 bloku zaczyna się zbiór **deskryptorów woluminu**, które mogą przybrać jedną z pięciu postaci:

- **Primary Volume Descriptor** jest obowiązkowym elementem zbioru. Zawiera m. in. identyfikatory systemu, woluminu, wydawcy i podmiotu przygotowującego dane, a także rozmiar bloku danych (który zamiast typowych 2048B w szczególnych przypadkach może wynosić wartość kolejnych wyższych potęg dwójki), liczbę bloków, datę stworzenia woluminu i przedawnienia informacji. Wymagany jest także wpis katalogu głównego.
- **Supplementary Volume Descriptor** posiada podobną strukturę, jest używany przez rozszerzenia.

- **Volume Partition Descriptor** w przypadku kiedy nośnik jest podzielony na partycje każda z nich musi posiadać taki deskryptor. Na jego podstawie można określić lokalizację i rozmiar partycji.
- **Boot Record** wykorzystywany przy płytach, z których można przeprowadzić rozruch komputera.
- **Volume Descriptor Set Terminator** oznacza koniec listy deskryptorów.

Katalogi są listami rekordów o zmiennej długości. W rekordach zapisane są dane takie jak wskaźnik na początek pliku, jego długość, atrybuty, numer płyty na której się znajduje, nazwa w formacie `nazwa.rozszerzenie;wersja` i przestrzeń do wykorzystania przez system operacyjny. Przewidziane jest miejsce (o rozmiarze zapisanym w rekordzie) na dodatkowe atrybuty, co pozwala rozszerzać możliwości standardu. Dla uproszczenia implementacji ograniczono głębokość zagnieżdżania katalogów do ośmiu poziomów.

W systemach komputerowych używa się dwóch sposobów na zapisywanie liczb w pamięci - big-endian i little-endian. Różnią się kolejnością zapisywania bajtów - w pierwszym zapis zaczyna się od najbardziej znaczącego bajtu, w drugim zaś od najmniej znaczącego. W ISO 9660, aby nie faworyzować żadnego z tych formatów, wykorzystuje się oba. Wszystkie pola liczbowe o wielkości większej niż bajt występują dwukrotnie - raz w little-endian, raz w big-endian.

1.2.5 ext2 - Second Extended File System

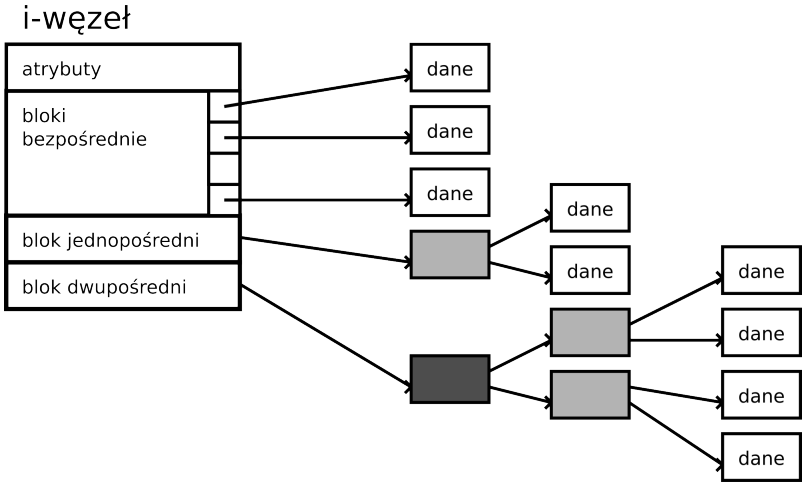
Linux w swoich pierwszych wydaniach korzystał z systemu plików odziedziczonego po Miniksie - minimalistycznym klonie Uniksa stworzonego dla celów edukacyjnych. Ten miał ograniczenia takie jak partycja o maksymalnym rozmiarze 64MB. By uniknąć tych ograniczeń w 1992 roku zaprojektowano Extended File System (ext), a niecały rok później zaprezentowano jego ulepszoną wersję - ext2. Specyfikacja tego systemu plików jest przedstawiona w [9].

Partycja ext2 na samym początku zawiera sektor rozruchowy, za którym znajdują się grupy bloków o strukturze przedstawionej w tabeli 1.6.

Super blok i deskryptory systemu plików zawierają podstawowe informacje niezbędne do poprawnego odczytania partycji takie jak np. rozmiar bloku. Każdy blok w grupie ma odzwierciedlenie w bitmapie, która określa czy jest on aktualnie wykorzystywany.

I-węzeł to struktura, która jednoznacznie opisuje plik, ale nie określa jego nazwy. Zawiera typowe atrybuty takie jak prawa dostępu, rozmiar, znaczniki czasowe czy położenie pliku na dysku. Położenie pierwszych dwunastu bloków jest zapisane bezpośrednio w węźle, kolejne korzystają z bloków pośrednich, czyli blok, którego adres zapisany jest w węźle zawiera listę bloków

Rysunek 1.1: I-węzeł systemu ext2



z danymi. Kiedy taką strukturę powiększymy o jeden poziom otrzymamy blok dwupośredni, dokładniej obrazuje to rysunek 1.1. I-węzły przechowywane są w tablicy dla każdej z grup bloków, a o tym czy są zajęte decyduje warość odpowiadającego im bitu w bitmapie i-węzłów.

W opisywanych wcześniej systemach plików wpis w katalogu dostarczał wszystkich informacji o pliku. Dzięki zastosowaniu i-węzłów katalog da się sprowadzić do listy par, gdyż wszystko co jest potrzebne do udostępnienia pliku użytkownikowi to jego nazwa i numer i-węzła.

Ext2 obsługuje dwa rodzaje dowiązań:

- **Dowiązania twarde** to zwykłe wpisy w katalogach - plik musi mieć przynajmniej jedno dowiązanie twarde, gdyż są one zliczane w i-węzłach. Gdy licznik ten osiągnie wartość 0 plik traktowany jest jako usunięty. Ze względu na swój charakter dowiązania twarde mogą występować tylko w obrębie jednej partycji.

- **Dowiązania sympoliczne** to pliki, które zawierają ścieżkę do innego pliku. System napotykając dowiązanie symboliczne rozpatruje operację na pliku ponownie, lecz z wykorzystaniem ścieżki zapisanej w dowiązaniu. Dzięki temu możliwe jest wskazanie na dowolny plik, niezależnie od partycji na której się znajduje.

1.2.6 NTFS - New Technology File System

NTFS został stworzony przez Microsoft, a jego pierwszą wersję wydano w 1993 roku wraz z Windows NT 3.1. Dzięki zastosowaniu go w aktualnych wydaniach systemu Windows jest najczęściej spotykanym systemem plików dla

Tabela 1.5: Struktura partycji ext2

Sektor rozruchowy
Pierwsza grupa bloków
Druga grupa bloków
...
N-ta grupa bloków

Tabela 1.6: Struktura grupy bloków ext2

Super blok
Deskryptory systemu plików
Bitmapa bloków
Bitmapa i-węzłów
Tablica i-węzłów
Bloki danych

komputerów domowych. W porównaniu z FAT NTFS jest znacznie bardziej złożony i funkcjonalny, a także pozbawiony wielu ograniczeń. Szczegółowy opis NTFS można znaleźć w [6].

Wolumin (partycja) NTFS jest podzielona na klastry o rozmiarze od 512 bajtów do 64 kilobajtów (w praktyce najczęściej spotyka się czterokilobajtowe klastry). W teorii maksymalny rozmiar woluminu (osiągalny dzięki 64 bitowej adresacji) to $(2^{64} - 1)n$, gdzie n to wielkość klastra, jednak aktualnie system Windows pozwala jedynie na 32 bitową adresację. Przy klastrze wielkości 64KB pozwala to stworzyć wolumin o rozmiarze $256TB - 64KB$.

Podstawą całego systemu jest **Master File Table** (MFT) - główna tablica plików zawierająca uporządkowane jednokilobajtowe rekordy opisujące pliki. Pierwsze 16 wpisow w MFT jest zarezerwowane dla plików metadanych, jednak aktualnie wykorzystywanych jest tylko 12 (opisanych w tabeli 1.7). Każdy wpis tej tablicy jest zbiorem atrybutów, których nagłówki zdefiniowane są w pliku *$AttrDef*. Kiedy wartość atrybutu (nazywana **strumieniem**) jest zbyt długa by zmieścić się bezpośrednio w rekordzie MFT istnieje możliwość przekształcenia go w atrybut nierezydentny, którego wartość przechowywana będzie w zaalokowanych w tym celu klastrach dysku. W przypadku gdy to ilość atrybutów jest problemem - wykorzystywana jest większa ilość wpisów w MFT.

Cechą odróżniającą NTFS od wcześniej opisanych systemów jest to, że sama treść pliku jest przechowywana jako strumień (domyślny strumień danych) - w identyczny sposób jak jego atrybuty. Takie podejście pozwala na

to, by plik posiadał dodatkowe strumienie danych (alternate data streams). Zdecydowano się na takie rozwiązanie dla kompatybilności z systemami firmy Apple, w których występują podobne mechanizmy.

Małe katalogi są przechowywane podobnie jak w innych systemach jako lista, jednak kiedy katalogi zawierają wiele wpisów takie podejście okazuje się mało wydajne. Aby zapobiec spadkowi wydajności przy poszukiwaniu konkretnego pliku wpisy w dużych katalogach są przechowywane w B+ drzewie. Kluczem w tej strukturze jest nazwa pliku.

NTFS ma także kilka dodatkowych funkcji takich jak opcjonalna kompresja plików (niewidoczna dla użytkownika), szyfrowanie za pośrednictwem sterownika EFS (Encryption File System) czy dziennikowanie - mechanizm zapamiętujący operacje dokonywane na systemie plików w celu łatwiejszego odtworzenia go w przypadku awarii.

Tabela 1.7: Master File Table (główna tablica plików) systemu NTFS

	Nazwa	**Rola pliku**
0	$Mft	Główna tablica plików - MFT także jest plikiem, co za tym idzie - pierwszym rekordem tablicy rekordem jest plik z jej własnym opisem.
1	$MftMirr	Kopia MFT utrzymywana na wypadek uszkodzenia oryginalnej. Dzięki temu ryzyko utraty danych jest mniejsze.
2	$LogFile	Informacje na temat operacji wykonywanych na systemie plików. Są one pomocne w przypadku awarii, ułatwiają przywracanie spójności systemowi plików.
3	$Volume	Dane na temat woluminu, np. jego etykieta
4	$AttrDef	Definicje atrybutów plików
5	. (kropka)	Katalog główny
6	$Bitmap	Mapa bitowa służąca do śledzenia wolnych klastrów
7	$Boot	Program rozruchowy, znajduje się w pierwszym klastrze woluminu
8	$BadClus	Lista klastrów, które znajdują się na uszkodzonych fragmentach dysku (badsectorach). Klastry z tej listy nie są używane podczas alokacji pamięci dla plików.
9	$Secure	Deskryptory praw dostępu. Są umieszczone w tym miejscu, by nie było konieczności ich wielokrotnego wpisywania gdy wiele plików posiada te same prawa.
10	$Upcase	W nazwach plików nie rozróżnia się wielkości liter - w tym pliku zawarta jest tablica konwersji wielkości liter dla różnych alfabetów.
11	$Extend	Katalog który może zawierać pliki z dodatkowymi metadanymi, np. limitami dyskowymi

Rozdział 2

Projekt BPFS

2.1 Cele projektu

Podczas projektowania systemu plików będącego przedmiotem tej pracy postanowiono, by częściowo oprzeć się na schematach znanych z innych systemów plików. Przede wszystkim BPFS do opisu pliku wykorzystuje i-węzły, co upodabnia go do uniksowych systemów plików, takich jak chociażby opisywany w poprzednim rozdziale ext2.

Cel postawiony przed BPFS to prostota zarówno od strony projektu, jak i implementacji, przy jednoczesnym zachowaniu niezbędnej funkcjonalności. Ma to umożliwić łatwość analizy, a co za tym idzie - możliwość wykorzystania jako prosty do przyswojenia przykład dla zainteresowanych sposobem działania systemów plików.

Z myślą o pamięciach USB zaplanowana została funkcja tworzenia macierzy partycji - system plików powinien wiele partycji fizycznych udostępniać jako jedną logiczną. Może to pozwolić na podzielenie danych na wiele nośników, a także uniemożliwić odtworzenie całej informacji w przypadku gdy któregoś z nośników zabraknie. Dla wzmocnienia tej funkcjonalności możliwe jest zastosowanie szyfrowania.

2.2 Struktura systemu plików

Podstawowa struktura partycji (obszaru pamięci trwałej zajmowanego przez BPFS) została przedstawiona w tabeli 2.1. Program zarządzający systemem plików na partycji nazywam sterownikiem.

2.2.1 Super blok

Super blok (nagłówek) BPFS znajduje się na samym początku partycji i zawiera podstawowe informacje na jej temat. Zawartość nagłówka została przedstawiona w tabeli 2.2.

Tabela 2.1: Struktura partycji BPFS

Super blok
Tablica i-węzłów
Dane menedżera wolnych bloków
Bloki danych

Tabela 2.2: Zawartość super bloku BPFS

bajty	**rozmiar**	**zawartość**
0 - 3	4	identyfikator systemu plików
4 - 7	4	rozmiar bloku
8 - 15	8	liczba bloków
16 - 17	2	rozmiar i-węzła
18 - 23	4	liczba i-węzłów

Pierwszym polem super bloku jest identyfikator systemu plików - łańcuch znaków o stałej zawartości ("BPFS"). Sterownik powinien podejmować próbę odczytania systemu plików tylko w przypadku, gdy identyfikator ma odpowiednią wartość.

Następnym polem jest rozmiar bloku. Możliwe wartości to wielokrotność rozmiaru sektora pamięci USB, który we współczesnych urządzeniach wynosi 512 bajtów.

Po nim następuje liczba bloków zapisana jako ośmiobajtowa liczba całkowita, co umożliwia stworzenie partycji o rozmiarze 2^{64} razy większym niż rozmiar bloku. Ostatnie dwa pola dotyczą własności i-węzłów i zostaną objaśnione w sekcji 2.2.2.

2.2.2 I-węzły

I-węzeł to struktura opisująca dokładnie jeden plik na partycji. Jego rozmiar jest określony w nagłówku i wynosi przynajmniej 32 bajty, gdyż tyle potrzeba by zawrzeć wszystkie wymagane pola (przedstawione w tabeli 2.3). Dodatkowo rozmiar i-węzła musi być dzielnikiem rozmiaru bloku - rozwiązanie to pozwala w całości wypełnić blok i-węzłami.

Pierwsze dwa bajty i-węzła to przestrzeń zarezerwowana dla przyszłych wersji BPFS na informacje o typie pliku oraz prawach dostępu do niego. Pierwszym istotnym polem i-węzła jest liczba dowiązań twardych, czyli liczba wpisów w katalogach, które wskazują na dany węzeł. Moment wyzerowania tego licznika jest równoznaczny z usunięciem pliku - i-węzeł, a także bloki zajmowane przez plik, którego był on opisem, powinny zostać zwolnione.

Tabela 2.3: I-węzeł BPFS, X oznacza rozmiar i-węzła w bajtach

bajty	**rozmiar**	**zawartość**
0 - 1	2	zarezerwowane dla przyszłego użycia
2 - 3	2	liczba dowiązań twardych
4 - 11	8	liczba bloków pliku
12 - 15	4	liczba wykorzystanych bajtów ostatniego bloku
16 - 23	6	wyrównanie, zarezerwowane dla przyszłego użycia
24 - $(X-9)$	8 na blok	bloki bezpośrednie
$(X-8)$ - $(X-1)$	8	blok pośredni

Następnie zapisana jest liczba bloków zajmowanych przez plik. Dzięki jej ośmiobajtowemu rozmiarowi istnieje możliwość, że plik zajmie całą dostępną przestrzeń partycji. Kiedy liczbę bloków pomnożymy przez rozmiar bloku dowiemy się jaką przestrzeń na nośniku zajmują dane pliku. Aby otrzymać dokładny rozmiar pliku należy uwzględnić fakt, że ostatni blok zazwyczaj nie jest w pełni zapełniony, a ilość bajtów, które są w nim faktycznie wykorzystane także została zapisana w i-węźle.

Jako minimalny rozmiar i-węzła przyjęte zostały 32 bajty. Dla wyrównania do tej wartości pozostawiono pustą sześciobajtową przestrzeń, za którą zaczyna się lista adresów bloków danych (bloków bezpośrednich) zakończona adresem bloku pośredniego. Blok pośredni także jest listą adresów bloków danych, którymi jest wypełniony prawie w całości, a jego ostatnie osiem bajtów wskazuje na kolejny blok pośredni. Konsekwencją wykorzystywania bloków pośrednich jest to, że przestrzeń dyskowa zajmowana przez plik nie ogranicza się tylko do tej wykorzystanej na dane - należy wziąć pod uwagę także stworzone bloki pośrednie.

Wszystkie i-węzły partycji są zapisane w wydzielonym im obszarze nazywanym **tablicą i-węzłów**. Liczba przechowywanych i-węzłów jest zadeklarowana w super bloku i wyznacza limit ilości plików, które mogą znaleźć się na partycji. Pierwszym elementem tablicy jest katalog główny.

2.2.3 Katalog

Katalog jest plikiem zawierającym listę wpisów, które są odnośnikami do innych plików. Każdy wpis zawiera trzy dane:

- **Długość wpisu** - wyrażona w bajtach.
- **Indeks i-węzła** w tablicy i-węzłów. Wskazywany i-węzeł jest opisem pliku, do którego odnosi się wpis.

- **Nazwa pliku** - ciąg znaków, dzięki któremu plik może zostać zidentyfikowany w sposób czytelny dla użytkownika.

Takie rozwiązanie sprawia, że nazwa nie jest zależna od samego pliku, a co za tym idzie - możliwe jest utworzenie kilku wpisów o różnych nazwach odnoszących się do tego samego pliku.

BPFS nie umożliwia tworzenia nowych katalogów, jedynym dostępnym katalogiem jest katalog główny.

2.2.4 Zarządzanie przestrzenią dyskową

Największy obszar partycji BPFS zajmuje przestrzeń przeznaczona na dane. Aby wydajnie wykorzystywać ten zasób należy ustalić sposób, który pozwoli na przydzielanie bloków dla plików, a także zadba o to, by miejsce zwolnione przez usunięte (lub zmniejszone) pliki mogło zostać ponownie wykorzystane.

Kiedy podjęta jest próba utworzenia nowego pliku sterownik powinien przeszukać tablicę i-węzłów w celu odnalezienia i-węzła o wyzerowanym liczniku dowiązań twardych. Następnie, aby do pliku zapisać dane, konieczne jest przydzielenie bloków dyskowych, czym zajmuje się podsystem nazwany **menedżerem wolnych bloków**.

Menedżer zarządza wolnymi blokami wykorzystując listę jednokierunkową. Pierwszy blok (blok kontrolny menedżera) za tablicą i-węzłów zawiera informacje umożliwiające lokalizację początku listy. W warstwie logicznej lista zawiera adresy bloków dyskowych w kolejności, w jakiej powinny zostać wykorzystane. Jej obrazem na dysku jest ciąg bloków wypełnionych adresami bloków. Ostatni z tych adresów wskazuje na kolejny element ciągu, pozostałe są wolnymi blokami (elementami listy). Po wyczerpaniu adresów z jednego z bloków ciągu kolejnym elementem listy jest jego własny adres.

Zapytanie menedżera o wolny blok powinno skutkować zwróceniem pierwszego elementu listy i zaktualizowaniem bloku kontrolnego. Lista wolnych bloków zajmuje coraz mniej miejsca wraz z coraz większym zapełnieniem partycji. Alternatywne do listy jednokierunkowej podejście, jakim jest mapa bitowa (wykorzystywana np. w ext2), spełniając podobne zadanie wymaga zarezerwowanej przestrzeni na dysku (jeden bit dla jednego bloku danych).

Kiedy plik jest usuwany wszystkie wykorzystywane przez niego bloki powinny być zgłoszone do menedżera jako zwolnione, co spowoduje dopisanie ich na początku listy bloków do wykorzystania. W razie konieczności na potrzeby listy zostanie zaalokowany nowy blok.

2.3 Macierze

BPFS umożliwia stworzenie macierzy dysków, która dla użytkownika będzie widoczna jako jedna partycja. Docelowo ma to pozwolić by dane zostały rozdzielone pomiędzy wiele pamięci USB. Takie pamięci mogą być przetrzymy-

wane w różnych miejscach, a do odtworzenia zapisanej informacji konieczne byłoby podłączenie każdej z nich do jednego komputera.

Działanie macierzy opiera się na prostym mechanizmie - mając d pamięci (indeksowanych od 0), w sytuacji gdy system plików chce otrzymać dostęp do k-tego bloku, powinien mu zostać udostępniony blok o numerze $(\lfloor \frac{k}{d} \rfloor + 1)$ z dysku o indeksie (k mod d). Przesunięcie numeru bloku o jeden wynika z przeznaczenia pierwszego bloku każdego z dysków na dodatkowe dane dotyczące szyfrowania, którego działanie zostanie przybliżone w sekcji 2.3.1.

2.3.1 Szyfrowanie

Do odczytania pełnych danych z macierzy konieczne jest uzyskanie dostępu do wszystkich nośników będących jej składowymi. Istnieje jednak ryzyko, że któryś z dysków zawiera kluczową porcję informacji, której tajność starano się uzyskać. Taka sytuacja zdecydowanie upraszcza kradzież danych, dlatego w celu wzmocnienia ich bezpieczeństwa należy zastosować szyfrowanie.

BPFS do szyfrowania wykorzystuje bardzo prosty algorytm - na każdym kolejnym 512 bajtowym fragmencie danych dokonywana jest operacja *xor* z kluczem o takim właśnie rozmiarze. Moc kryptograficzna takiego rozwiązania nie jest zbyt wielka, ale jego prostota stanowi atut jeśli chodzi o łatwość w analizie, która jest jednym z celów całego projektu. Każdy z nośników ma swój własny klucz szyfrujący zapisany na innym dysku macierzy. Przy liczbie pamięci wynoszącej d klucz dla nośnika o indeksie i znajduje się na nośniku o indeksie $(i + 1)$ mod d.

Rozdział 3

Implementacja BPFS

3.1 Wykorzystane narzędzia

BPFS został zaimplementowany dla systemu operacyjnego Linux. Ze względu na wsparcie dla programowania obiektowego i kompatybilność z językiem C do zaprogramowania BPFS został wybrany język **C++** (w standardzie z 2011 roku). Korzystanie z programowania obiektowego pozytywnie wpływa na czytelność kodu, a kompatybilność z językiem C pozwala na wykorzystanie biblioteki **FUSE** (opisanej w sekcji 3.1.1).

Dla uproszczenia procesu budowania programów wykorzystano narzędzie **CMake**. Dzięki niemu możliwe jest podzielenie projektu na mniejsze składowe (własne biblioteki, wiele programów wykonywalnych) i zarządzanie ich kompilacją. Nie bez znaczenia jest fakt, że środowisko programistyczne wykorzystywane podczas tworzenia BPFS (**Qt Creator**) posiada wsparcie dla CMake.

3.1.1 FUSE - Filesystem in Userspace

FUSE jest biblioteką pozwalającą na tworzenie w pełni funkcjonalnych systemów plików w przestrzeni użytkownika. Oznacza to, że BPFS funkcjonuje bez wprowadzania zmian w jądrze systemu operacyjnego. Dokumentację FUSE można odnaleźć na oficjalnej stronie internetowej [10].

Wszystkie systemy plików obsługiwane przez jądro korzystają z **wirtualnego systemu plików** (VFS, virtual file system). VFS obsługuje wywołania systemowe dotyczące plików (np. `open()`, `read()`) w taki sposób, by program uruchomiony przez użytkownika działał niezależnie od systemu plików w jakim sformatowany jest nośnik na którym pracuje. Aby system plików mógł być obsługiwany przez jądro musi implementować zbiór funkcji i struktur umożliwiających komunikację z wirtualnym systemem plików.

FUSE działa dzięki modułowi, który jest zdolny do komunikacji z VFS i udostępnia interfejs, który może zostać wykorzystany w programie uruchamianym przez użytkownika. Takie rozwiązanie pozwala, by kod systemu

plików omijał ograniczenia narzucane przez jądro, takie jak chociażby brak dostępu do standardowych bibliotek C. Najprostszy przykładowy system plików napisany z wykorzystaniem FUSE (który po zamontowaniu zawiera jeden plik tekstowy) wymaga zaimplementowania jedynie czterech funkcji[1].

3.2 Struktura i realizowane zadania

Kod implementacji BPFS jest podzielony na cztery moduły:

- **core** - główna biblioteka, w której znajduje się cała logika opisana w rozdziale 2.
- **file-access** - biblioteka odpowiedzialna za dostęp do nośnika danych poprzez jego plik specjalny z katalogu `/dev/`.
- **manager** - aplikacja BPFS Manager
- **fuse-wrapper** - aplikacja BPFS Driver

3.2.1 Dostęp do nośnika danych

System operacyjny Linux udostępnia nośniki danych jako pliki specjalne - urządzenia blokowe. Dzięki temu użytkownik może odczytać i modyfikować zawartość nośnika w sposób niemal identyczny jak każdy inny plik (nie jest możliwa jedynie zmiana wielkości pliku, gdyż musi ona odpowiadać dostępnej w urządzeniu pamięci).

3.2.2 BPFS Manager

BPFS Manager to program, którego zadaniem jest formatowanie partycji do systemu plików BPFS i tworzenie macierzy nośników. Korzystanie z programu umożliwia prosty interfejs tekstowy.

BPFS Manager wymaga, by jako argument podczas uruchamiania podać ścieżkę do partycji (urządzenia blokowego) na którym ma operować. Komenda

```
./manager /dev/sdb1
```

pozwoli na wykorzystanie partycji `/dev/sdb1` w BPFS Manager. Podczas korzystania z programu istotne jest posiadanie prawa do zapisu i odczytu wskazanego urządzenia blokowego.

Do formatowania partycji służy komenda `mkfs`, po której wywołaniu użytkownik zostaje zapytany o rozmiar bloku, rozmiar i-węzła oraz liczbę i-węzłów:

[1] http://fuse.sourceforge.net/helloworld.html

```
$ ./manager /dev/sdb1
/dev/sdb1::???> mkfs
Block size: 1024
Inode size: 256
Number of inodes: 32
```

Podobnie wygląda proces tworzenia macierzy, który zostaje rozpoczęty po skorzystaniu z komendy `mkmatrix`. Różnicą w stosunku do `mkfs` jest to, że w pierwszej kolejności program pyta o ilość dysków, z których będzie składać się macierz. Jako pierwszy element macierzy traktowane jest urządzenie blokowe wskazane podczas uruchamiania programu, kolejne muszą zostać podane przez użytkownika:

```
$ ./manager /dev/sdb1
/dev/sdb1::???> mkmatrix
Matrix size: 3
Device 0: /dev/sdb1
Device 1: /dev/sdc1
Device 2: /dev/sdd1
Block size: 1024
Inode size: 256
Number of inodes: 32
```

3.2.3 BPFS Driver

BPFS Driver to program zarządzający pracą systemu plików BPFS. To za jego pośrednictwem, dzięki wykorzystaniu biblioteki FUSE, partycja BPFS jest montowana w systemie. BPFS Driver korzysta z trybu debugowania FUSE, dzięki czemu widoczny jest jasny opis tego, jakie operacje są wykonywane na systemie plików w danym momencie.

Aby zamontować partycję stworzoną w BPFS Manager poprzez komendę `mkfs` należy wywołać:

```
./fuse-wrapper <urządzenie_blokowe> <punkt_montowania>
```

Aby zamontować macierz dysków należy skorzystać z komendy:

```
./fuse-wrapper matrix <u0> <u1> ... <uN> <punkt_montowania>
```

gdzie `<u0> <u1> ... <uN>` to bezwzględnie ścieżki do urządzeń blokowych, z których składa się macierz. Warto zauważyć, że istotna jest ich kolejność - powinna być identyczna do tej, którą wykorzystano podczas tworzenia macierzy. Przyjmując `/mnt/bpfs` jako punkt montowania, systemy plików podane jako przykłady w sekcji 3.2.2 mogą zostać zamontowane w systemie komendami:

```
./fuse-wrapper /dev/sdb1 /mnt/bpfs
./fuse-wrapper matrix /dev/sdb1 /dev/sdc1 /dev/sdd1 /mnt/bpfs
```

Zamontowanie partycji BPFS pozwala na dostęp do zasobów na niej zgromadzonych zarówno z poziomu linii poleceń, jak i przy wykorzystaniu graficznych eksploratorów takich jak Nautilus czy Dolphin.

Rozdział 4

Możliwości rozwoju BPFS

4.1 Rozszerzenie funkcjonalności

Jak już zostało wspomniane wcześniej - BPFS cechuje prostota ułatwiająca analizę działania. Dlatego też podczas projektowania zrezygnowano z wprowadzenia pewnych funkcjonalności, których obecność we współczesnym systemie plików wydaje się być oczywista. Nie oznacza to jednak, że owe funkcjonalności nie są możliwe do zaimplementowania. Co więcej, zarówno projekt jak i jego implementacja były tworzone z myślą o możliwości dodania niektórych z nich.

W tabeli 2.3 można zauważyć, że pierwsze dwa bajty i-węzła zostały zarezerwowane dla przyszłego użycia. Jest to miejsce, które może zostać przeznaczone na informacje o prawach dostępu - do ich przechowania w formacie typowym dla Uniksa wystarczy 9 bitów (prawo do zapisu, odczytu i wykonania dla właściciela pliku, grupy właściciela i dla reszty użytkowników systemu). Pozostałe 7 bitów może być wykorzystane do określenia rodzaju pliku.

Jednym z rodzajów pliku, które można by wyróżnić jest katalog. Aktualnie jedynym katalogiem w BPFS jest katalog główny, jednak już teraz jest on zaimplementowany w sposób, który pozwala traktować go tak, jak każdy inny plik. Aktualny stan w znaczny sposób ułatwia dodanie możliwości tworzenia nowych katalogów, dzięki czemu BPFS może stać się hierarchicznym systemem plików.

Zastosowanie macierzy powoduje, że fragmenty plików znajdują się na różnych nośnikach, zaś powszechne dzisiaj procesory wielordzeniowe pozwalają na wykonywanie wielu operacji równocześnie. Rozsądnym wydaje się wykorzystanie równoległego odczytywania danych z wielu nośników, aby przyspieszyć działanie systemu plików. Niestety zachowanie programu wykonywanego na wielu rdzeniach jest dużo bardziej nieprzewidywalne od programu sekwencyjnego. Dzieje się tak przez współdzielenie niektórych zasobów

przez wiele wątków, podczas gdy kolejność, w jakiej poszczególne wątki będą się do nich odwoływać może być różna przy każdym uruchomieniu. BPFS dla przyspieszenia wielokrotnego odczytywania tych samych obszarów nośnika tymczasowo zapisuje niektóre dane w pamięci komputera. Sposób w jaki to dokonano uniemożliwia równoległe wykonywanie programu - odczytywane dane stają się zniekształcone. W przypadku dalszego rozwijania BPFS powinno się poświęcić temu problemowi szczególną uwagę.

4.2 Moduł dla VFS

Biblioteka FUSE zdecydowanie upraszcza tworzenie systemu plików, jednak niesie za sobą pewne ograniczenia. FUSE jest dodatkową warstwą pośredniczącą między systemem a urządzeniem podczas przekazywania danych, co negatywnie wpływa na wydajność działania. Zintegrowanie z jądrem dodatkowo umożliwia wykorzystanie systemu plików do rozruchu samego Linuksa.

W [11] możemy przeczytać jakie ograniczenia niesie za sobą pisanie kodu jądra. Przede wszystkim wykorzystywanym tam językiem jest C w standardzie GNU. Oznacza to, że tworząc moduł wirtualnego systemu plików dla BPFS konieczne byłoby napisanie od nowa większości kodu, gdyż istniejący korzysta z wielu cech C++ niedostępnych w C. Niemniej jednak stworzenie takiego modułu byłoby wartościowym doświadczeniem i jest warte rozważenia.

4.3 Konwersja na inne systemy operacyjne

Większość kodu BPFS (cała biblioteka `core` oraz program BPFS Manager) została napisana w niezależnym od platformy standardzie C++11. Biblioteka `file-access` jest związana z platformą, choć użycie jej w innych systemach uniksowych nie powinno stanowić problemu. Możliwość skorzystania z istniejącego kodu BPFS Driver zależy przede wszystkim od dostępności FUSE lub podobnych bibliotek w docelowym systemie operacyjnym.

FUSE dla Linuksa jest jedyną oficjalnie rozwijaną wersją biblioteki, jednak istnieją także wersje przystosowane do działania na innych systemach operacyjnych. Spośród nich warto wymienić *FUSE for FreeBSD* i *OSXFUSE* (dla OS X firmy Apple).

BPFS może zostać przekonwertowany także na systemy operacyjne, dla których nie istnieje FUSE stosując podobne biblioteki. Podobną funkcjonalność dla Windows dostarcza aktualnie nie rozwijana biblioteka *Dokan*. Istnieje także komercyjne rozwiązanie - *Callback File System* firmy Eldos. W systemie operacyjnym NetBSD możliwe jest skorzystanie z biblioteki *puffs*.

Bibliografia

[1] Andrew S. Tanenbaum, *Modern Operating Systems*. Prentice Hall, Second Edition, 2001.

[2] R. C. Daley, P. G. Neumann, *A General-Purpose File System For Secondary Storage*. http://www.multicians.org/fjcc4.html, internet

[3] Tom Van Vleck, *Multics Glossary*, http://www.multicians.org/mgloss.html, internet

[4] Andrew S. Tanenbaum, *Systemy operacyjne*. Helion, Wydanie III, Gliwice, 2012.

[5] Tim Paterson, *An Inside Look at MS-DOS*, http://www.patersontech.com/dos/byte-inside-look.aspx, internet

[6] Mark Russinovich, David A. Solomon, Alex Ionescu *Windows Internals* Microsoft Press, Sixth edition, 2012.

[7] *Detailed Explanation of the FAT Boot Sector*, DEW Associates Corporation, http://www.dewassoc.com/kbase/hard_drives/boot_sector.htm, internet

[8] Standard ECMA-119, *Volume and File Structure of CDROM fot Information Interchange*

[9] Rémy Card, Theodore Ts'o, Stephen Tweedie *Design and Implementation of the Second Extended Filesystem*, http://e2fsprogs.sourceforge.net/ext2intro.html, internet

[10] http://fuse.sourceforge.net, internet

[11] Robert Love *Jądro Linuksa. Przewodnik programisty*, Helion, Gliwice, 2014.