Wirtualny hybrydowy system plików

Sebastian Madajczyk
sebam@mat.umk.pl

1 września 2010

Spis treści

Streszczenie

W większości współczesnych systemów operacyjnych najpopularniejszym modelem organizacji systemu plików jest drzewo katalogów. Koncepcja hierarchicznego systemu plików w postaci drzewa katalogów została opublikowana już w roku 1965 i wdrożona niedługo potem w systemie operacyjnym Multics. Dzisiaj, gdy pamięć masowa jest tania i nawet na maszynach użytku domowego znajdują się ogromne ilości różnorodnych danych, rozwiązania sprzed prawie pół wieku trudno uznawać za wygodne. Największym problemem jest efektywne odszukanie informacji, które za pomocą narzędzi wykonujących rekursywne przejście po drzewie katalogów jest nieakceptowalne ze względu na wydajność.

Szybki dostęp do informacji jest elementem krytycznym dla wydajnej pracy z systemem informatycznym. Użytkownicy przyzwyczajeni do korzystania z wyszukiwarek internetowych i bardzo szybkich odpowiedzi na zapytania o dane w sieci globalnej oczekują przynajmniej tego samego od narzędzi przeszukujących lokalne zasoby informacji na ich stacji roboczej, sieciowym serwerze plików, czy na serwerze ogólnego przeznaczenia wykorzystywanym równolegle przez kilkuset użytkowników na uczelni. Odpowiedzią na nowe coraz większe zapotrzebowania są zdobywające coraz większą popularność narzędzia indeksujące, które działając w tle tworzą indeksy wyszukiwania (Beagle, Apple Spootlight, Microsoft Windows Search, Meta tracker itp.). Niektóre z nich są już dojrzałymi projektami zintegrowanymi z popopularnymi systemami operacyjnymi. Zawierają one najczęściej interfejs graficzny, oraz konsolowy, który z reguły słabo integruje się ze starszymi aplikacjami innymi niż powłoka systemowa. Przyczyną tego stanu rzeczy jest definiowanie własnych, niekompatybilnych z istniejącymi aplikacjami interfejsów. W niniejszej pracy staram się rozważyć stworzenie modelu interfejsu, który mimo że będzie korzystał z interfejsu hierarchicznego systemu plików, tym samym dając nowe funkcjonalności istniejącym aplikacjom, to oferował będzie duże możliwości wyszukiwania.

Niniejsza praca ma na celu zaproponować nowy projekt systemu plików, który będzie stanowił hybrydę istniejących rozwiązań. Rozważana będzie koncepcja, która łączy w sobie cechy modelu hierarchicznego oraz semantycznego, przy czym model semantyczny zawierać będzie zarówno podejście klasyczne zaprezentowane przez twórców koncepcji semantycznego systemu plików - wykorzystujące atrybuty plików jak i koncepcję etykiet rozbudowaną o możliwość systematyzowania. Większość udogodnień nowej koncepcji zakłada wykorzystanie istniejącego interfejsu systemu plików, dzięki czemu integracja ze starszymi aplikacjami nastąpi w sposób nieinwazyjny.

Rozdział 1

Wstęp

System komputerowy umożliwia zapis informacji na nośnikach różnego typu. W celu zwiększenia wygody w użytkowaniu tych nośników system operacyjny dostarcza abstrakcyjny widok logiczny dla danych na nośniku oraz określony zbiór operacji pozwalających zmieniać ów widok poprzez zmianę danych. Rozwiązanie to pozwala aplikacjom na abstrahowanie od własności technicznych urządzeń przechowywania i odczytu danych. Podstawowym pojęciem które pozwala na tworzenie widoków jest plik, który stanowi nazwaną, logiczną jednostkę powiązanych ze sobą informacji. Koncepcja pliku stwarza na aplikacjach wymóg - dane mogą być odczytywane i zapisywane na nośniku tylko za pomocą odpowiednich operacji na pliku. Inaczej rzecz ujmując jedynym sposobem na zmianę lub odczyt zawartości urządzenia przechowywania danych jest skorzystanie z istniejącego, lub utworzenie nowego pliku. Nie każdy nośnik pozwala jednak na zmianę zawartości plików po ich pierwszym zapisaniu. Nośnik może mieć też inne ograniczenia, które system operacyjny musi uwzględnić przy oferowaniu interfejsu zmiany stanu nośnika użytkownikowi.

Systemy operacyjne są tworami wyjątkowo złożonymi, co wiąże się z koniecznością istnienia ogromnej ilości logicznie różnych zbiorów informacji - plików. Projektanci pierwszych systemów operacyjnych stanęli więc przed dylematem - w jaki sposób ułatwić aplikacjom i użytkownikom systemu korzystanie z urządzeń magazynowania danych. Sama abstrakcja pliku nie była wystarczająca i należało wprowadzić usystematyzowanie dostępu do plików, czyli stworzyć strukturę łatwą do zrozumienia, pozwalającą na intuicyjną organizację plików za pomocą zdefiniowanego zbioru operacji, przy jednoczesnym uwzględnieniu ograniczeń sprzętowych. Struktura taka, wraz ze zbiorem operacji, które pozwalają na jej zmianę nazywa się systemem plików. Przedstawienie pewnego sposobu systematyzowania plików jest kluczem do niniejszej pracy. Inaczej mówiąc praca ta podejmuje problem zmiany abstrakcyjnego widoku i operacji skojarzonych z tym widokiem czyli zmiany semantyki systemy plików.

System plików jest ważną częścią systemu operacyjnego. W przypadku systemu UNIX jest fundamentem na którym tworzony jest cały system operacyjny [RT74]. Od czasów powstania pierwszych implementacji systemu UNIX najpopularniejszym modelem systematyzacji plików aż do dzisiaj jest model hierarchiczny, który zakłada istnienie wielopoziomowego drzewa katalogów. Idea hierarchicznego systemu plików powstała nawet o wiele wcześniej niż pierwsze implementacje UNIX-a i została włączona do systemu operacyjnego Multics [DN65]. Mimo że, hierarczhiczne systemy plików ewoluowały i są dzisiaj tworami o wiele bardziej złożonymi niż u swoich początków, to warta zanotowania jest niezmienność semantyki hierarchicznych systemów plików z punktu widzenia użytkownika. Innym ważnym faktem jest niesłabnąca popularność tego rozwiązania - dzisiaj zdecydowana większość systemów plików wykorzystuje strukturę wielopoziomowego drzewa katalogów, a wszelkie innowacje współczesnych systemów plików dotyczą przede wszystkim cech poszczególnych implementacji takich jak:

- rozmieszenie danych na dysku
- zabezpieczanie danych przed niespójnościami
- elementy sieciowości

a więc cech, które nie wpływają prawie w żaden sposób na interakcję użytkownika z systemem plików. W pracy postaram się także uzasadnić tezę, że hierarchiczny system plików stał się mało praktyczny na przestrzeni lat mając na uwadzę przede wszystkim sposób interakcji użytkownika z nim i dlatego warto rozważyć stworzenie zmienionego interfejsu, który ułatwi pracę użytkownika z tym bardzo ważnym elementem systemu operacyjnego.

Większość użytkowników radzi sobie na swój sposób z niedoskonałościami najpopularniejszego modelu (nawet niedoskonałości tych sobie nie uświadamiając) dzięki doświadczeniom zdobytym w ich użytkowaniu. Użytkownicy wiedzą jaki sposób rozmieszenia danych w drzewie katalogów jest dla nich najbardziej wygodny. Przeważnie pliki organizowane są poprzez wykonanie odpowiedniej kategoryzacji, co wydaje się praktyką odpowiednią dla tegoż modelu. Każdy sposób kategoryzacji danych jest jednak sztuką kompromisu, o czym, mam nadzieję, będę potrafił przekonać czytelnika, w przypadku gdyby świadomości tego narzuconego kompromisu jeszcze nie miał.

Celami niniejszej pracy jest zaproponowanie alternatywnego podejścia do sposobu interakcji z systemem plików, który będzie oferował pewne cechy semantycznego systemu plików - wykorzystanie zbioru par {klucz, wartość} do określenia pewnych cech pliku oraz pewne cechy systemu opartego o etykietowanie plików. Koncepcja tu przedstawiona zakłada, że każdemu plikowi można nadać wiele etykiet, które na dodatek mogą być umiejscawiane w pewnej hierarchii. Najistotniejsze założenia pracy można streścić w kilku podpunktach:

- Projekt logiczny języka, którego konstrukcja będzie pozwolała na dobrą integrację z interfejsem systemu plików i tym samym będzie zapewniała wsparcie dla istniejących aplikacji, bez jakielkowiek ingerencji w ich kod
- Projekt logiczny systemu plików
- Szczegóły i analiza implementacji

Jako, że zagadnienie, którego dotyczy praca jest bardzo obszerne, to zamieściłem w pracy także propozycje rozbudowy implementacji systemu plików o dodatkowe funkcje, które z powodu ograniczonych ram czasowych były niemożliwe do zrealizowania.

1.1 Motywacja

Główną motywacją dla tego projektu jest chęć zwiększenia wydajności pracy z systemem plików poprzez zmianę semantyki podstawowych operacji na nim.

W popularnych hierarchicznych systemach plików plik z reguły wystepuje tylko w jednym miejscu. Użytkownik ma możliwość tworzenia twardych i symbolicznych linków, ale w przypadku hierarchicznej struktury katalogów wykorzystanie mechanizmu linkowania dla usprawnienia organizacji danych jest praktyką mało powszechną, prawdopodobnie dlatego, że niewygodną i czasochłonną.

Konstrukcja hierarchicznego systemu plików zakłada dostęp do plików poprzez ścieżkę wykorzystującą ustalony separator elementów ścieżki (w przypadku systemów UNIX - ukośnik „/", w przypadku MS Windows - odwrotny ukośnik „\". Elementami ścieżki są kolejne węzły drzewa katalogów, czyli nazwy katalogów oraz nazwy plików. Większość systemów operacyjnych stowarzysza katalog roboczy z wykonywanym procesem tak jak ma to miejsce w systemach UNIX [Ste02]. Zwykle dostęp do tego katalogu uzyskujemy poprzez odwołanie się do ścieżki „.". Inną nazwą katalogu o specjalnym znaczeniu jest „.." i oznacza katalog o jeden stopień wyżej w hierarchii katalogów względem katalogu roboczego. Wprowadzenie specjalnych nazw katalogów pozwala twórcom aplikacji na abstrahowanie od miejsca położenia plików wykonywalnych w drzewie katalogów i zwalnia ich z przewidywania pełnej ścieżki bezwględnej do wymaganych plików które są potrzebne do poprawnego działania aplikacji. Ścieżki względne (a więc takie, które zaczynają się od nazw specjalnych) są prostym mechanizmem zapewniającym uproszczenie dostępu do plików zależnych.

Z punktu widzenia użytkownika, który chce uzyskać dostęp do pliku z reguły konieczna jest znajomość bezwględnej ścieżki dostępu - a więc wszystkich kolejnych składowych ścieżki dostępu (tzw. wpisów katalogowych (ang.

directory entry) - czyli nazw katalogów oraz plików) z zachowaniem określonej kolejności lub też ścieżki względnej względem katalogu w którym się znajduje. Może się zdarzyć, że względna ścieżka dostępu składa się z większej ilości elementów i wymaga większej wiedzy o strukturze katalogów, niż bezwględna ścieżka dostępu. W obydwu przypadkach trudno oczekiwać aby użytkownik posiadał precyzyjną wiedzę o położeniu wszystkich plików w drzewie katalogów. Użytkownicy będąc świadomymi specyfiki hierarchicznego modelu systemu plików organizują dane w sposób najbardziej dla siebie odpowiedni z uwzględnieniem ograniczeń tegoż modelu. Rozpatrzmy pewne szczególne scenariusze organizacji danych w hierarchicznym systemie plików. Załóżmy, że osoba będąca użytkownikiem postanowiła stworzyć strukturę katalogów, która odzwierciedla podział plików ze względu na różne cechy plików.

1. Podział dokumentów (załóżmy, że użytkownik jest uczniem liceum, a wszystkie pliki jakie będzie rozmieszczał mają rozszerzenia: *.pdf, *.odt albo *.doc)
    - pliki może rozdzielić na katalogi względem roku nauki
        - 1 klasa, 2 klasa, 3 klasa, 4 klasa
    - lub względem dziedzin, których dotyczą
        - matematyka, informatyka, fizyka, biologia
2. Podział plików graficznych (załóżmy, że są to tylko zdjęcia wykonane przez użytkownika)
    - miejsce w którym dane zdjęcie zostało zrobione:
        - Paryż, Singapur, Bangkok, Buenos Aires, Perth, Dom Rodziców, Dom Ewy
    - data związana ze zdjęciem
        - 20 czerwca 2007, 2 maja 1993, 17 lipca 2008
3. Podział plików muzycznych (załóżmy, że są to tylko pliki z rozszerzeniem mp3):
    - epoka (muzyczna dekada):
        - lata 90., lata 80., lata 70., lata 50., bardzo stare
    - gatunek muzyczny:
        - klasyka, jazz, hip hop, rock

Każdy z tych układów katalogów został stworzony przez użytkownika najprawdopodobniej w sposób ręczny, a sposób organizacji został przez niego wymyślony i w zależności od jego konsekwencji wdrożony. W czasie tworzenia struktury katalogowej pewien sposób organizacji danych wydał się mu

najbardziej adekwatny do późniejszego sposobu ich użytkowania. Załóżmy jednak, że chce on zaprezentować komuś z najbliższych zdjęcia z imprez rodzinnych. Jeśli wybraną metodą organizacji danych jest ta z datą jako atrybutem decydującym o przyporządkowaniu do miejsca w drzewie katalogów, to odszukanie wszystkich plików, które dotyczą spotkań rodzinnych może wiązać się z koniecznością przejrzenia wszystkich podkatalogów dla poszczególnych dat. Użytkownik systemu hierarchicznego ma wówczas do wyboru przynajmniej kilka scenariuszy reorganizacji struktury katalogowej, która pomogłaby mu pliki wyszukać szybciej w przypadku, gdyby takie zadanie musiał wykonać poraz kolejny. Rozważmy możliwości reorganizacji danych w systemie plików, jakie może wykorzystać użytkownik.

1. W każdym katalogu z datą w nazwie utworzyć podkatalogi, które będą zawierały informacje o miejscu zrobienia zdjęcia.

2. Utworzyć katalogi o nazwach, które są nośnikiem informacji o miejscu zrobienia zdjęcia. Następnie wewnątrz tych katalogów stworzyć podkatalogi zawierające informacje o dacie zrobienia zdjęcia. Odszukanie wszystkich zdjęć kojarzonych z określonym miejscem stanie się łatwiejsze, ale odszukanie wszystkich zdjęć wykonanych konkretnego dnia może wymagać użycia dodatkowych narzędzi wyszukiwania

3. Utworzyć nową strukturę katalogową, która będzie istniała równolegle do tej, która już istnieje w drzewie katalogów (dla każdej metody organizacji poddrzewa utworzyć katalog i umieścić w nim utworzone poddrzewo), następnie utworzyć odpowiednie dowiązania symboliczne lub twarde wg nowej zasady kategoryzacji w katalogach tego poddrzewa. W ten sposób każde kryterium wyszukiwania będzie miało swoje poddrzewo wyszukiwania

4. Może także skorzystać z oprogramowania do indeksowania zdjęć, które albo stworzy pewną abstrakcję na dane (np. poprzez ekstrakcję metadanych EXIF) widzianą tylko wewnątrz danego programu, lub przeorganizuje położenie plików na dysku w sposób podobny do tego propowanego w poprzednim podpunkcie. Aplikacje tego typu zwykle mają możliwość przyporządkowywania etykiet do zdjęć, więc wyszukanie mogłoby odbywać się również za pomocą porównania etykiet.

Pierwsza metoda jest bardzo mało elastyczna i w przypadku zmiany kryteriów wyszukiwania najprawdopodobniej wydłuży proces wyszukiwania plików (z powodu powiększenia się drzewa katalogów).
Drugie rozwiązanie również jest mało elastyczne i cierpi na podobną przypadłość jak to pierwsze - komplikuje i powiększa drzewo katalogów czyniąc wyszukiwanie szybszym tylko w szczególnym przypadku.
Trzecie rozwiązanie nie ma wad poprzedników, ponieważ dla każdego kryterium wyszukiwania odpowiednie poddrzewo może być równie proste

w swojej budowie jak pierwsza kategoryzacja jaką wykonał użytkownik. Ręczne utworzenie takich dodatkowych poddrzew wypełnionych dowiązaniami wymaga jednak bardzo dużego nakładu pracy, co czyni rozwiązanie zupełnie niepraktycznym.

Rozwiązaniem najwygodniejszym pod względem szybkości wyszukiwania i czasu potrzebnego do reorganizacji danych jest rozwiązanie czwarte, wymaga ono jednak dodatkowego oprogramowania. Przykładami aplikacji, które oferują takie funkcje są: Digikam[1], F-Spot[2], Google Picasa[3], iPhoto[4] i wiele innych. Pozwalają na ekstrakcji metadanych z plików, oraz dają możliwość etykietowania plików przez użytkownika. Cechą wspólną tych aplikacji jest kiepska integracja z oprogramowaniem zewnętrznym, oraz ograniczenie tylko do plików graficznych. Gdybyśmy chcieli uzyskać podobną funkcjonalność automatycznej kategoryzacji dla innych typów plików, to będziemy zmuszeni użyć innego oprogramowania. Dla różnych typów plików używalibyśmy różnych aplikacji o odmiennych interfejsach, które z reguły nie są zintegrowane z resztą systemu operacyjnego tak dobrze jak system plików, a stanowią tylko abstrakcję na istniejący hierarchiczny system plików.

Warte odnotowania jest, że przy każdym z przykładów poczyniliśmy dla uproszenia, że pliki mają pewne własności takie jak niewielki zbiór obsługiwanych formatów. W rzeczywistości użytkownik systemu plików może być grafikiem, który korzysta z grafik wektorowych i rastrowych, tworzy animacje gif, oraz jest też zapalonym podróżnikiem, który systematycznie odwiedza różne miejsca na Ziemi. Łatwo sobie wyobrazić, że w zależności od sposobu użycia plików graficznych nasz grafik będzie potrzebował zupełnie różnych metod kategoryzacji plików.

[1] http://digikam.kde.org
[2] http://f-spot.gnome.org
[3] http://www.picasa.google.com
[4] http://iphoto.apple.com

1.2 Cele

Istnieje wiele systemów, które wspomagają indeksowanie i dzięki temu szybkie wyszukiwanie plików na dysku. Część z nich jest wbudowana w większe aplikacje, które służą do przeglądania zawartości plików i indeksowania zwykle tylko jednej klasy plików czyniąc je bardzo słabo zintegrowanymi z systemem plików. Część ma z założenia możliwości indeksowania plików różnego rodzaju - takich jak pliki multimedialne, dokumenty, odnośniki internetowe, historie rozmów w komunikatorze, pliki poczty i wykonuje indeksowanie w tle cały czas podczas pracy z systemem operacyjnym. Istnieją także projekty systemów plików stanowiących alternatywę dla hierarchicznych systemów plików. Przed przejściem do omówienia poszczególnych rozwiązań warto zastanowić się czego należy oczekiwać od wydajnego rozwiązania indeksowania danych, a co jednocześnie powinno być priorytetowymi założeniami tej pracy.

Najważniejszymi celami jakie postawił sobie autor pracy względem projektowanego systemu są:

1. Zaprojektowanie interfejsu systemu plików, który będzie pozwalał na organizację plików w bardziej naturalny i intuicyjny sposób niż system hierarczhiczny

2. Uwzględnienie problemów kompatybilności z istniejącymi aplikacjami, a zatem projekt języka, który byłby wyrażalny poprzez wyrażenia ścieżkowe (**odnośnik do definicji**) i pozwalałby na wykonanie zapytań wyszukujących różnego typu - implementował by operatory logiczne „AND”, „OR”, „NOT” oraz niektóre operatory algebry relacji - takie jak sortowanie, agregacja,

3. Zaprojektowanie interfejsu dla narzędzia automatycznej ekstrakcji metadanych z plików

Istnieją także cele, które nie są tak istotne jak powyższe, ale również zostały rozważone w pracy:

1. projekt rozszerzonego interfejsu GUI, który zwiększyłby integrację systemu plików z podsystemem graficznym

2. projekt podpowiedzi / wskazówek dla użytkownika systemu plików

3. propozycje rozwiązania problemów skalowalności (możliwości udostępniania zasobów sieciowo, obsługa bardzo dużych ilości danych)

1.3 Kompromisy

Jak zawsze w przypadku implementacji nowych pomysłów powstają pewne granice, których położenie jest determinowane kompromisem pomiędzy

wydajnością, a możliwościami funkcjonalnymi implementowanego systemu. Mimo, że założeniem tego projektu nie jest tworzenie aplikacji klasy produkcyjnej a tylko implementacji, która miałaby stanowić demonstrację możliwości takiego systemu, to autor miał na uwadze zagadnienia takie jak: zużycie pamięci operacyjnej, wydajność oraz skalowalność. W ten sposób koncepcje, które są z powodów wydajnościowych nie sprawdziłyby się w dużych systemach plików obsługujących miliony plików zostały odrzucone. Mimo, że w projekcie pozostały tylko funkcjonalności, które mogą być implementowane wydajnie, to w tak wczesnym stadium rozwoju implementacji nie może być mowy o dużych optymalizacjach kodu i wszystko co nie działa ekstremalnie niewydajnie jest do zaakceptowania.

1.4 Rozwiązanie

Projekt jest rozwiązaniem częściowym, które nie zakłada całkowitej rezygnacji z hierarchicznych systemów plików. Zamiast całkowitego odejścia od sprawdzonych rozwiązań proponuje się tu oswajanie użytkowników z nowymi podejściami do systemów plików. Projekt zakłada stworzenie abstrakcji, która w odróżnieniu do innych istniejących abstrakcji posiada podobny interfejs do hierarchicznego systemu plików i tym samym pozwala zapewnić mu kompatybilność z istniejącymi aplikacjami bez ingerencji w ich kod. Autor jest zdania, że najlepszą drogą zmian dla tak kluczowego elementu systemu operacyjnego jakim jest system plików jest droga ewolucji.

Implementacja wykorzystuje bibliotekę Fuse umożliwiającą tworzenie systemów plików działających w kontekście procesu a nie kontekście jądra. Docelowo system plików powinien działać w kontekście jądra - zarówno ze względu na wydajność jak i problemy spójności danych. (**Rozszerzyć**)

Rozdział 2

Krótka historia systemów plików

2.1 Wczesne systemy plików

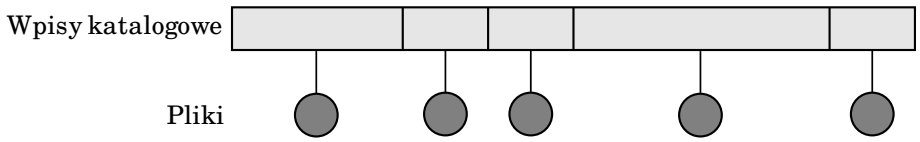
(**Jakiś wstęp, że architektura Von Neumana zakłada istnienie pamięci głównej, oraz „drugiej“, czyli masowej, i że pamięć masowa jest konieczna i trzeba ją ustruktyzować.**)

2.1.1 Płaskie systemy plików - katalog jednopoziomowy

Systemy plików istnieją od 1964 roku wraz z powstaniem technologii DECTtape [bit09]. Dostarczała ona pamięć masową dla systemów operacyjnych firmy Digital Equipment Corporation zaczynając od PDP-6. Pierwszy system plików pozwolił na utworzenie 27 plików w katalogu, przy czym każdy z nich składał się z jednego lub więcej bloków wielkości 256 osiemnastobitowych słów. Każdy plik tegoż systemu identyfikowany był nazwą składającą się z dwóch 6 znakowych podnazw. Jest to najprostsza wykorzystywana struktura katalogowa, która zakłada, że wszystkie pliki znajdują się w jednym katalogu głównym, i są rozróżnialne dzięki unikatowej nazwie. Może zaskoczyć czytelnika, że koncepcja jednokatalogowych systemów plików była wykorzystywana w systemach operacyjnych obsługujących duże jednostki obliczeniowe aż do sierpnia 1985 roku gdy firma Apple Computers zastąpiła Macintosh File System (MFS) systemem HFS (Hierarchical File System). Mimo że, MFS był systemem z katalogiem jednopoziomowym, to interfejs graficzny dostarczany wraz z systemem operacyjnym tworzył jednak iluzję drzewa katalogów [SG96].

2.1.2 Katalog dwupoziomowy

Rozwinięciem koncepcji katalogu jednopoziomowego był katalog dwupoziomowy [SG96]. Wprowadzono tutaj pojęcie katalogu plików użytkownika



Rysunek 2.1: Płaski system plików

(ang. *user file directory* - ***UFD***). Każdy katalog plików użytkownika jest katalogiem jednopoziomowym, dzięki czemu problem z nazywaniem w taki sam sposób plików przez wielu użytkowników został wyeliminowany. Każdy UFD jest podkatalogiem głównego katalogu plików (ang. *master file directory* - ***MFD***). Użytkownik zalogowany do systemu, a więc posiadający swój UFD, w przypadku próby otwarcia pliku o określonej nazwie inicjuje akcję przeszukania swojego katalogu domowego przez system operacyjny. W przypadku, gdy procedura wyszukania pliku w UFD zakończy się niepowodzeniem, to wyszukiwanie jest kontynuowane w katalogach określonych przez odpowiednią zmienną środowiskową. Zdolność tworzenia nowych UFD przypada tylko użytkownikom z uprawnieniami administratorskimi. (**odnaleźć systemy operacyjne i sensowne publikacje na ten temat**)

2.1.3 Hierarchiczne systemy plików

Hierarchiczne systemy plików to inaczej katalogi o strukturach drzewiastych, które są kolejnym rozwinięciem katalogu jednopoziomowego. W 1972 r. 6 wersja Unix-wprowadziła nowy system plików o nazwie V6FS, który był ewolucją systemu plików dla Multicsa, stworzonego w 1969 roku. Opracowanie przestawiające strukturę drzewiastą jako rozwiązanie dla pamięci rozszerzonej zostało opisane już w roku 1965 jako owoc współpracy MIT z laboratoriami Bella [DN65]. Od tego momentu hierarchiczna struktura stała się najpopularniejszą metodą organizacji plików i oprócz systemów uniksowych została zaadaptowana także przez inne popularne systemy operacyjne w postaci systemu plików FAT firmy Microsoft i HFS firmy Apple. Unix udostępniał cztery podstawowe abstrakcje związane z systemem plików: pliki, katalogi, i-węzły oraz punkty montowania dzięki którym wiele systemów plików tworzy jedną spójną strukturę drzewiastą. Katalogi są szczególnymi plikami zawierającymi informacje o innych plikach znajdujących się w danym katalogu. I-węzły zawierają informacje na temat pliku: typ, bity praw dostępu, rozmiar, wskaźniki do bloków danych itp. Katalog zawiera tylko dwie informacje o pliku: numer i-węzła, oraz nazwę pliku [Ste02].

Krokiem milowym w technologii systemów plików stało się wprowadzenie dziennikowania, które poraz pierwszy zostało włączone do systemu plików ODS-2 systemu operacyjnego OpenVMS w 1979. System ten oferował tylko dziennikowanie operacji wykonanych na metadanych dotyczących plików (takich jak właściciel pliku, grupa, uprawnienia, informacje o typie pliku).

Podsystem dziennikowania polega na zapisywaniu przyszłych zmian w systemie plików w odpowiednim dzienniku. Wykonanie tego kroku ma na celu zabezpieczenie spójności danych w przypadku awarii. Podczas kolejnego podmonowywania systemu plików nie ma potrzeby sprawdzania spójności całego systemu plików, wystarczy przejrzeć dziennik logów od odpowiedniego miejsca i porównać go ze stanem systemu plików.

Udoskonalenie to zostało zapożyczone z systemów zarządzania bazami danych, gdzie integralność danych jest rzeczą kluczową. Istnieją także systemy, które oferują pełne księgowanie zdarzeń (w ten sposób, aby dane użytkownika były w pełni zabezpieczone w razie awarii, co wymaga dwukrotnego zapisu danych na nośniku), jednak ze względów wydajnościowych nie są popularne. Technologia używana w ReiserFS4 o nazwie (ang. *wandering logs*) pomaga zredukować narzut związany z dwukrotnym zapisem danych. Pierwszym komercyjnym zastosowaniem dziennikowania w systemach plików był VxFS od firmy Vertias, ale pierwszym powszechnie używanym stał się dopiero NTFS, który był używany zarówno do zastosowań komercyjnych jak i zastosowań domowych.

Innym udoskonaleniem systemów plików były wielokrotne/alternatywne strumienie danych (ang. alternate data streams), które zakłada skojarzenie więcej niż jednego zbioru danych binarnych do jednej nazwy pliku. Wiele współczesnych systemów plików pozwala na teoretycznie nieskończoną liczbę alternatywnych strumieni danych skojarzonych z jednym plikiem, co czasami jest nazywane jako rozszerzone atrybuty (ang. extended attributes).

Stosunkowo nowym udogodnieniem oferowanym przez systemu plików są rozszerzone listy kontroli dostępu (Access Control Lists - ACL's), które pozwalają na bardziej szczegółowe określenie uprawnień użytkowników, niż było to możliwe za pomocą tradycyjnego podziału uprawnień na użytkownika, grupę i innych użytkowników. Taka bardziej rozbudowana forma zarządzania uprawnieniami użytkowników na poziomie systemu plików została wprowadzona przez firmę Novell w systemie operacyjnym NetWare i systemie plików NWFS w 1985 roku.

2.2 Współczesne systemy plików

Współczesne systemy plików obsługiwane przez systemy uniksowe wykorzystują abstrakcję wirtualnego systemu plików (ang. *virtual filesystem - VFS*). Jest to podsystem jądra implmentujący interfejs systemu plików udostępniany programom przestrzeni użytkownika. VFS zapewnia nie tylko współistnienie systemów plików ale i ich współpracę. Pozwala na wykorzystywanie standardowych uniksowych wywołań systemowych do odczytu i zapisu różnych systemów plików, zakładanych na różnych nośnikach [Lov04]. Rozwiązanie to zostało stworzone dla systemów hierarchicznych i każdy system plików montowany do drzewa katalogów przez system uniksowy musi

dostosować się do niego - jest to fakt bardzo istotny dla wszystkich twórców systemów plików (także tych systemów plików, które nie są hierarchiczne). VFS jest dla systemu operacyjnego rozwiązaniem, które bardzo upraszcza zarządzanie zróżnicowanymi systemami plików.

2.2.1 Implementacje

Jednym z najbardziej popularnych systemów plików w systemie operacyjnym GNU/Linux jest Ext3, który jest kontynuacją Second Extended File System (ext2) i dodaje do niego dziennikowanie. Ext3 oprócz dziennikowania zdarzeń oferuje także Listy kontrolne dostepu (ACL), miękkie i twarde dowiązania, oraz rozszerzone atrybuty (EA - extended attributes).

Kolejnym zyskującym szybko na popularności systemem plików jest ReiserFS od firmy Namesys. Posiada on wszystkie cechy Ext3, oraz zwiększoną skalowalność (oprócz poprawy wydajności większe mogą być też rozmiary partycji oraz rozmiary plików). Trwają prace nad reiser4, który już zyskał na sporej popularności. Jednak ten system plików ciągle jeszcze nie został włączony do głównej gałęzi jądra zarządzanej przez Linusa Torvaldsa.

Zdecydowana większość implementacji systemu plików skupiona jest jednak na poprawie algorytmów klastrowania, oraz zwiększaniu możliwości i wydajności w pracy wielodostępnej (czego przykładami mogą być implementacje GFS, GPFS, OCFS oraz Lustre. Dla przykładu system plików od firmy Google (GFS) został stworzony dla środowisk rozproszonych. System ten zakłada dostęp do wszystkich zasobów poprzez serwer główny zawierający metadane (a więc dane o danych) takie jak położenie rzeczywistych plików na serwerach pomocniczych. GFS musi uwzględniać dużą awaryjność serwerów pomocniczych i zabezpieczać się przed nią wykorzystując redundację danych.

2.3 Przyszłość systemów plików

Trudno przewidzieć odległą przyszłość tak dynamicznej dziedziny jaką jest informatyka. Wydaje się jednak, że systemy plików, a także oferowane przez nie funkcje muszą ulec modyfikacjom. Sygnały ku temu dają już od jakiegoś czasu wiądocy producenci systemów operacyjnych tacy jak: Microsoft z inicjatywą WinFS, oraz Apple Computers ze zintegrowanym narzędziem Apple Spootlight. Rówież w GNU/Linux istnieje coraz większa grupa aplikacji indeksujących, oraz nakładek tworzących alternatywne metody organizacji danych, można tu wymienić chociażby: GNOME Storage project, Tagsistant, DBFS dla KDE.

Każde z tych rozwiązań jest jednak ściśle powiązane z jednym systemem operacyjnym i nie wydaje się aby mogły stać się rozwiązaniem bardziej ogólnym. Większość z nich wykorzystuje relacyjną bazę danych jako tzw. backend. Hans Reiser w swoim opracowaniu wskazuje, że model relacyjny nie

jest dobrym rozwiązaniem jeśli chodzi o semantyczne systemy zarządzania informacjami.

System plików zawsze tworzony jest pod konkretne zastosowanie o czym mogliśmy się przekonać omawiając systemy takie jak GFS. Prawie zawsze o wyjątkowości niektórych z nich decyduje jedna cecha spośród wielu jakie mają systemy plików. Od jednego systemu możemy oczekiwać się szczególnej troski o integralność danych (wówczas jednak procedura tworzenia lub modyfikacji plików będzie trwała dłużej), wtedy przykład systemu plików opracowanego przez firmę Google nie sprawdzi się najlepiej. Jeszcze czegoś innego będzie wymagać od systemów plików dla urządzeń mobilnych - prawdopodobnie będzie to wymóg ograniczenia do minimum użycia przestrzeni dyskowej.

Rozdział 3

Kategoryzacja danych

Historia organizacji plików z uwzględnieniem ich semantyki nie jest historią najnowszą jak na dynamikę rozwoju gałęzi nauki jaką jest informatyka. W tym rozdziale po krótce zaprezentowane zostaną projekty, które miały największe znaczenie dla rozwoju. Nie wszystkie z wymienionych projektów są implementacjami systemu plików. Część z nich jest lepiej lub gorzej zintegrowanymi z systemem operacyjnym aplikacjami. Wszystkie mają jednak wspólny cel - przyspieszyć proces wyszukiwania plików.

3.1 Magazyny dokumentów

Magazyny dokumentów oprócz indeksowania plików oferują także magazyn danych, a więc inaczej mówiąc są pojemnikiem dla danych. Jeśli magazyn wykorzystuje relacyjną bazę danych, to pliki mogą być w niej reprezentowane jako wartości na polu typu BLOB.

3.1.1 Implementacje

Windows Future Storage - tak brzmiała pełna nazwa kodowa relacyjnego systemu zarządzania i magazynowania danych rozwijanego przez firmę Microsoft. Celem projektu było dostarczenie warstwy abstrakcji na SQL Server powyżej systemu plików NTFS, dzięki której będzie możliwe szybsze wyszukiwanie plików zgodnie z kryteriami wyrażonymi przez użytkownika w języku OPath. Projekt po raz pierwszy został zademonstrowany w 2003 na „Professional Developers Conference“ jako magazyn danych dla nadchodzącej wersji systemu Microsoft Windows (Vista), a materiał filmowy w momencie tworzenia pracy był dostępny w internecie pod nazwą IWish. Został stworzony aby dostarczyć możliwość zarządzania danymi dobrze ustrukturyzowalnymi, semi-ustrukturyzowalnymi i zupełnjie nieustrukturyzowalnymi. Projekt został jednak zaniechany w czerwcu 2006 roku i nie włączony ostatecznie do wersji Vista systemu Windows. Wdrożenie tego systemu, lub

odpowiadającego jemu stoi pod znakiem zapytania.

GNOME Storage był projektem, który miał podobne założenia jak WinFS. Przede wszystkim głównym celem była możliwość szybkiego wyszukiwania plików z uwzględnieniem metadanych. Aplikacja dostarczała parsowanie wyrażeń języka naturalnego i nie była systemem plików, a raczej integralną częścią środowiska graficznego. Mimo udanych początków projekt został zawieszony w październiku 2004 roku. Założeniem projektu było całkowite zwolnienie użytkownika z konieczności ręcznej kategoryzacji plików. GNOME Storage dostarczał filtrów do ekstrakcji metadanych z plików, ale ilość oferowanych filtrów nie była aż tak okazała jak w przypadku innych omawianych implementacji (np. LibFerris o którym później).

System operacyjny **BeOS** powstał w 1991 roku jako projekt firmy Be Inc., specjalnie na potrzeby komputera BeBox. System został stworzony z myślą o zastosowaniach multimedialnych i posiadał wiele unikatowych własności względem innych istniejących w tym czasie systemów operacyjnych. System plików BFS z którego korzystał OS pozwalał na tworzenie wielu zdefiniowanych przez użytkownika atrybutów w celu identyfikacji obiektów w systemie plików. Bynajmniej nie ograniczło się to tylko do plików ale i struktur zamkniętych w pliku np. wiadomości e-mail. Użytkownik miał możliwość utworzenia żywych widoków, które aktualizują wyniki wyszukiwania jak tylko skojarzone z nim indeksy się zmienią. Widoki te zachowują się jak normalne katalogi, które zawierają pliki spełniające kryteria określone przez użytkownika.

Tagsistant jest semantycznym systemem plików, który został poraz pierwszy opublikowany w licpu 2007. Wykorzystuje bibliotekę SQLite, oraz bibliotekę FUSE i jest projektem międzyplatformowym. W związku z zależnościami systemu obsługuje systemy z rodziny GNU/Linux, BSD, oraz OS-X. Twórcy systemu plików zakładają, że system plików jest najbardziej uniwersalnym interfejsem dla wszystkich programów użytkowych, dlatego najlepszym narzędziem do szybkiego wyszukiwania w systemie plików jest również system plików. W lutym 2008 roku dodane zostały funkcjonalności takie jak rozpoznawanie kiedy etykieta (tag) reprezentuje zbiór będący podzbiorem zbioru reprezentowanego przez inną etykietę. System rozpoznaje też etykiety semantycznie identyczne[1] Problemem tej implementacji jest niska wydajność, co jest efektem użycia biblioteki SQLite, która nieposiada wsparcia dla indeksów.

IBM Rationale Clearcase ™ jest to system kontroli wersji cieszący się dużą popularnością wśród korporacji na całym świecie. System ten wykorzystuje ideę dynamicznych widoków specyfikowanych poprzez pliki konfiguracyjne (configspec). Poza głównym zadaniem - wersjonowaniem przechowywanych obiektów (każdy z obiektów posiada swoją historię wersji), posiada

[1] semantyczna identyczność oznacza, że zbiory stowarzyszone z dwoma etykietami zawierają te same elementy

również możliwość nadawania etykiet, a także przechowywania wielu innych metadanych na temat plików. Ciekawa z punktu widzenia tej pracy jest relatywnie niska wydajność tego systemu kontroli wersji, mimo że system działa w przestrzeni jądra.

3.2 Indeksy metadanowe

W odróżnieniu do magazynu dokumentów, indeks metadanowy nie pozwala na utrzymywanie zawartości plików wewnątrz magazynu danych z którego korzysta aplikacja indeksująca. Zamiast tego tworzone są i utrzymywane różne elementy metadanych kojarzonych z plikami. Indeksy te mogą być odświeżane automatycznie za użyciem narzędzi automatycznie wykonujących taką notyfikację w wypadku występienia jakiegoś zdarzenia, które najczęściej wbudowane są w jądro systemu operacyjnego. Alternatywnie indeks może być odbudowywany cyklicznie poprzez przechodzenie po **drzewie katalogów** - w tym przypadku należy mieć na uwadze kompromis między narażeniem się na niespójności indeksu z aktualnym stanem obserwowanego systemu plików, a narzutem wydajnościowym powodowanym przez wykonanie przejścia po drzewie katalogów.

Jak łatwo się domyślić - rozwiązania tego typu nie mogą być samodzielnym systemem plików, ale prawdopodobnie mogą być narzędziami bez interfejsu systemu pliku, lub z takim interfejsem, ale zrealizowany jako wirtualny system plików.

Apple Spotlight to narzędzie zaprezentowane w kwietniu 2005 roku wraz z systemem operacyjnym Mac OS X ver. 10.4 którego jest integralną częścią. Narzędzie pozwala na indeksowanie także innych obiektów niż pliki - takich jak np. e-maile, ustawienia systemowe, teksty z komunikatorów. Spotlight oprócz indeksowania na bazie metadanych wykonuje też indeksowanie pełnotekstowe przeszukując dokumenty. Odświeżanie indeksu następuje w sposób automatyczny, gdy tylko plik jest tworzony, lub modyfikowany, eliminując narzut związany z cyklicznym przechodzeniem po drzewie katalogów. Aplikacja wyposażona jest we wtyczki zdolne do ekstrakcji metadanych, oraz wykonywanie wspomnianego wcześniej wyszukiwania pełnotekstowego. Oprócz tego firma Apple udostępniła także API pozwalające twórcą nowych formatów plików tworzyć podobne wtyczki do ekstrakcji metadanych. Spotlight jest narzędziem graficznym, które pozwala na przeglądanie zawartości plików bez potrzeby ich otwierania w narzędziach dedykowanych.

Beagle jest narzędziem do przeszukiwania zasobów stacji roboczej, które zostało przedstawione w kwietniu 2004 roku. Do indeksowanie pełnotekstowego wykorzystuje silnik Apache Lucene. Indeksuje nie tylko pliki, ale i inne obiekty w systemie plików. Jest narzędziem dość popularnym w niektórych dystrybucjach systemu GNU/Linux.

Projekt **Meta Tracker** jest związany ze środowiskiem graficznym GNO-

ME. W odróżnieniu od Beagle, który jest stworzony przy pomocy technologii Mono, tracker jest napisany w języku C. Meta Tracker jest bardziej zgodny z wytycznymi freedesktop.org, tym samym prawdopodobnie ma większe szanse stać się projektem przyszłościowych niż Beagle.

Google Destkop to aplikacja, która oferuje sposób wyszukiwania podobny do stylu wyszukiwarki internetowej firmy Google. Aplikacja jest w stanie indeksować wiadomości e-mail, pliki tekstowe, zdjęcia, dzienniki czatów itp. Firma Google jako jedyna zadbała, aby jej rozwiązanie było międzyplatformowe. Systemami operacyjnymi wspieranymi przez Google Desktop są: systemy firm: Microsoft i Apple (integruje się z aplikacją Spotlight) oraz GNU/Linux. Pierwsza opublikowana wersja datowana jest na październik 2004, wersje dla systemów innych niż Microsoft Windows zostały udostępnione w roku 2007.

LibFerris jest to biblioteka dynamiczna, która swoją dojrzałość implementacyjną zdobywa już od 2001 roku. Celem jej twórców jest dostarczenie łatwego dostępu do metadanych różnych obiektów powstałych w trakcie aktywności użytkownika systemu operacyjnego. Ilość typów wspieranych obiektów z których biblioteka potrafi wyłuskać dane budzi respekt:

- ekwiwalent metadanych o plikach, które oferuje polecenie systemowe lstat
- wartości funkcji skrótu na pliku (md5, sha1)
- serializowaną wartość pliku w postaci XML
- schemat oraz zawartość relacyjnych oraz słownikowych bazy danych
- wsparcie dla Linux EA ((ang. *Extended Attributes*))
- tagi ID3 dla plików MP3, OGG
- metadane obiektów graficznych: jpg, png, gif, djvu itd. (wszystkie formaty wspierane przez ImageMagick oraz imlib2)
- metadane obiektów filmowych (mpeg2)
- metadane obiektów dźwiękowych (dolby AC3)
- wiele innych

Wszystkie te metadane dostępne są za użyciem interfejsu C++, który wykorzystuje znane i dobrze rozumiane abstrakcje STL-owe takie jak iteratory i strumienie IOStream. Biblioteka wspiera bezpośrednio serializację do formatu XML, oraz rozszerzenie XSLT pozwalające przetwarzać plik XML zgodnie ze zdefiniowanymi regułami. Dostępne są podpięcia ((ang. *bindings*)) do języków python, perl oraz OCaml. (libferrisui.so ())

Możliwe jest również montowanie obiektów i dostęp do odczytu a w niektórych przypadkach również do zapisu za pomocą abstrakcji wirtualnego systemu plików, co wykorzystuje bibliotekę fuse.

- treści dostępne poprzez protokoły:
  - http
  - ftp
  - ssh
  - LDAP
- archiwa:
  - rpm files
  - tar files
  - XML
  - mbox
  - bazy danych:
    - berkeley db4
    - mysql [2]
    - sleepycats dbxml (przejęta przez Oracle)
    - tdb (Trivial Database)[3])
    - edb (EnterpriseDB) [4]
    - gdbm [5]
- obiekty systemu operacyjnego:
  - sysv IPC (pamięć dzielona)
  - video dvd
  - sockets

3.3 Koncepcje teoretyczne semantycznych systemów plików

Wiele rzeczywistych systemów plików, lub innych rozwiązań indeksowania danych zostało stworzonych na podstawie eksperymentów opisanych w publikacjach na temat semantycznych systemów organizacji danych. Krótki przegląd najbardziej znanych publikacji został przedstawiony poniżej.

[2]http://www.mysql.com
[3]http://tdb.samba.org
[4]http://www.enterprisedb.com
[5]http://www.vivtek.com/gdbm

Semantic File System jest to pierwsza koncepcja semantycznego systemu plików, która została przedstawiona przez grupę zajmującą się rozwojem Systemów operacyjnych na MIT w 1991 roku [GJSO91].

Omawiana praca prezentuje podstawowe idee oraz motywację do sematnczynej organizacji systemu plików, a także korzyści jakie taki sposób organizacji systemu plików by przyniósł. Publikacja zawiera opis implementacji prototypu takiego systemu plików w oparciu o NFS. System jest wirtualnym systemem plików, ponieważ koncepcja zakłada tworzenie katalogów w momencie wysłania zapytania o konkretny katalog systemu plików. Nazwy katalogów występujące w wyrażeniu ścieżkowym są kluczami wyszukiwania. Za pomocą odpowiednich zapytań możemy dowiedzieć sie jakie są możliwe wartości kluczy (wynik przedstawiający możliwe wartości kluczy również przygotowany jest dopiero w razie potrzeby). Gdy już mamy parę klucz i wartość, to możemy uzyskać dostęp do plików, które mają przyporządkowanie klucz - wartość takie o jakie zapytaliśmy.

Przykład zapytania podany przez autorów:

```
1  $ cd /sfs/exports:/lookup_fault
2  $ ls -F
3  virtdir_query.c@       virtdir_query.o@
4  $ cd ext:/c
5  $ ls -F
6  virtdir_query.c@
7  $
```

Przykład demonstruje dużą siłę tego typu zapytań. (**Dopisać**)

Projekt **pStore** to koncepcja teoretyczna opublikowana przez firmę Hewlett Packard nie doczekała się implementacji. Opisywany nie jest system plików, a pewien magazyn danych, którego głównym założeniem jest elastyczność i rozszerzalność. Proponowanym modelem danych jest RDF (ang. *Resource Description Framework*). Wiele cech proponowanego rozwiązania przewyższa możliwościami systemy oparte o relacyjne bazy danych. Przykładem jest lepsza zdolność do dynamicznej ewolucji schematu bazy danych. Jest również prosty i lekki, ponieważ wiele aplikacji nie wymaga założeń ACID, System ten zakłada również wersjonowanie plików jako integralną część systemu plików. Ważną cechą jest też wykorzystywanie skojarzeń - asocjacji pomiędzy przechowywanymi obiektami, tak aby system plików, bądź też inny magazyn plików bardziej przypominał sposób działania ludzkiego mózgu, niż systemy istniejące obecnie. Autorzy zwracają jednak uwagę na trudność implementacji takiego systemu, przy jednoczesnym przekonaniu o jego ogromnych zaletach.

Rozdział 4

Projekt VHFS

4.1 Wstęp

Projekt VHFS w swojej pierwszej, ograniczonej wersji, ma działać w systemie operacyjnym GNU/Linux. Autor zakłada wykorzystywanie Fuse - modułu jądra Linux, który pozwala na utworzenie systemu plików w przestrzeni użytkownika. Decyzja o wykorzystaniu tego podejścia, jak wiele decyzji projektowych ma swoje wady i zalety. Do zalet systemów plików w przestrzeni użytkownika oraz biblioteki libFuse, która tego typu podejście umożliwia należą:

- łatwiejsze naprawianie błędów czasu wykonania programu w przypadku programów działających w przestrzeni użytkownika, niż tych działających w przestrzeni jądra
- wykorzystanie zrębu [GHJV95] (ang. *framework*) do budowania systemu plików, zamiast tworzenia mechanizmów od zera
- możliwość prototypowania w językach skryptowych z wykorzystaniem już istniejącego podpięcia (ang. *binding*)
- montowanie wielu systemów plików przez jednego użytkownika, lub wielu użytkowników jednocześnie, z możliwością łatwego ograniczenia dostępu do swoich magazynów metadanych

Podejście to ma jednak również wady:

- gorsza wydajność z powodu dodatkowego poziomu abstrakcji, oraz wykorzystania języka skryptowego z późnym wiązaniem nazw
- zależność od innego systemu plików

Dostęp do metadanych będzie realizowany za pomocą bazy danych. W najprostszym modelu, który jest implementowany w tej chwili, wykorzystane jest obiektowo-relacyjne przyporządkowanie (ang. *object relational mapping*), które potrafi korzystać z wielu baz danych. Wybór padł na SQLite

ponieważ jest to baza danych natywnie wspierana przez standardową bibliotekę języka python. Język ten został wykorzystany do implementacji prototypu, ponieważ posiada wszystkie potrzebne podpięcia i jest dojrzałym językiem o eleganckiej i prostej składni. W przypadku powodzenia prototypu autor rozważa implementację podejścia w postaci modułu jądra, który byłby implementowany w C++. Bardzo prawdopodobnym byłaby też rezygnacja z relacyjnej bazy danych. Możliwe rozwiązania dla składu danych to: wbudowana obiektowa baza danych, bądź implementacja własna składu danych. Warto byłoby się przyjrzeć potencjalnym korzyścią do których mogłoby prowadzić magazynowanie także zawartości plików, a więc nie tylko metadanych. Póki co autor zdecydował się jednak ograniczyć rozważania do wirtualnego systemu plików opartego na linkach symbolicznych.

4.2 Wymagania funkcjonalne

Wymagania funkcjonalne zdefiniowane zostaną za pomocą szeregu przypadków użycia, najpierw jednak należy zdefiniować zbiór pojęć, które będą wykorzystywane podczas specyfikacji wymagań.

4.2.1 Definicje

- **identyfikator** - identyfikatorami nazywamy:
  - aliasy do pliku
  - nazwy:
    - etykiety
    - atrybuty
    - nazwy przestrzeni nazw
    - widoki
    - metody
- **identyfikator globalny** - identyfikator, który jest widoczny z poziomu każdego wyrażenia
- **plik** (ang. *file*)
  W wirtualnym systemie plików jest to obiekt trzymający referencję do pliku, który fizycznie znajduje się poza tym systemem plików. W wynikach zapytań wirtualnego systemu plików pliki reprezentowane są jako linki symboliczne.
- **etykieta** (ang. *tag*)
  Jest to ciąg znaków (nazwa), który można skojarzyć z określonym plikiem. Ciąg ten spełnia warunki:

  - jest zbudowany ze znaków alfabetu łacińskiego, cyfr, oraz znaku podkreślenia
  - nie zaczyna się od znaku podkreślenia
  - nie wystepują w nim wielkie litery
  - ilość znaków należy do przedziału $[2, 32]$
  - jest różny od wszystkich istniejących identyfikatorów globalnych (wielkość liter jest rozróżniana)

  Ponadto etykiety zorganizowane są w struktury drzewiaste, co powoduje, że przydatne stają się dodatkowe definicje:

  - **etykieta pierwszego rzędu**
    Jest to etykieta, która jest korzeniem dla swojego drzewa etykiet. Etykieta ta nie posiada etykiety rodzica, co powoduje, że drzew etykiet jest dokładnie tyle ile etykiet pierwszego rzędu
  - **podetykieta**
    Jest to etykieta, która posiada rodzica w drzewiastej strukturze etykiet, np. `A` jest podetykietą `B - A` jest dzieckiem `B`
  - **nadetykieta**
    Etykieta, która jest rodzicem w drzewiastej strukturze etykiet, np. `A` jest nadetykietą `B - A` jest rodzicem `B`

  Uwaga
  Jeśli plik ma przyporządkowaną jakąkolwiek podetykietę etykiety `A`, oznacza to, że ma również przyporządkowaną etykietę `A`

- **atrybut** (ang. *attribute*) - posiada kilka cech:
  - oznacza cechę pliku
  - posiada nazwę (N) oraz wartość (V) określonego typu (T)
  - nazwa atrybutu (N) jest ściśle powiązana z typem wartości, co tworzy klasę atrybutów $(C =< N, T >)$
  - klas atrybutów jest dokładnie tyle ile nazw atrybutów, a więc klasy można identyfikować poprzez nazwy
  - każdy plik posiada przynajmniej 2(?) atrybuty (instancje odpowiednich klas): `id` oraz `name`

- **operator**
  - **operator logiczny** - jeden z operatorów: `and, or, not, AND, OR`
  - **operator arytmetyczny** - jeden z operatorów: `+ - * / %`
  - **operator algebraiczny** - jeden z operatorów: (**wypisać listę**)

- **wyrażenie ścieżkowe** - ciąg znaków, który jest poprawny syntaktycznie względem zdefiniowanej gramatyki dla ścieżek i posiada semantykę zależną od operacji na systemie plików której podano ścieżkę
- **widok** (ang. *view*) - link do wyrażenia ścieżkowego - stanowi skrót dla wyrażenia

Podstawowy zestaw wymagań

Projektowany system plików powinien wspierać:

1. dodawanie i usuwanie plików do indeksu.
2. tworzenie i usuwanie etykiet oraz podetykiet.
3. przypisywanie i usuwanie etykiet plikom.
4. deklarację typów wartości dla atrybutów.
5. nadawanie i usuwanie atrybutów plikom.
6. obsługę atrybutów plików standardowego systemu plików takich jak:
    - data modyfikacji, dostępu, utworzenia.
    - rozmiar
    - prawa dostępu
7. wyszukiwanie plików za pomocą wyrażeń ścieżkowych.
8. dobrą integrację z istniejącym interfejsem systemu plików
    - Zapytanie nie powinno być tworzone "na raz", lecz stopniowo. Zgrubnie mówiąc każdy dodatkowy separator katalogów oznacza początek kolejnego kroku. Dzięki tego typu podejściu łatwiej będzie uniknąć błędów.
9. tworzenie dynamicznych widoków

Całkowita spójność danych jest pożądana ale nie jest wymagana, tzn. może się zdarzyć, że linki symboliczne będą prowadzić do plików, które nie istnieją, albo zmieniły swoją nazwę. Ważne, aby system był w stanie tego typu niespójności naprawiać automatycznie.

Wspólne założenia dla wszystkich przypadków użycia

Aktorami dla zdefiniowanych przypadków użycia są:

- użytkownik systemu plików (może być to aplikacja korzystająca z wirtualnego systemu plików)
- system operacyjny
- wirtualny system plików

Założenia początkowe:

- punktem montowania wirtualnego systemu plików jest */vhfs*
- użytkownik ma prawo do odczytu i zapisu
- użytkownik nie jest rootem (dla uproszczenia)

4.2.2 Domyślne przestrzenie nazw

Nazwy przestrzeni nazw są identyfikatorami pierwszego rzędu, co oznacza, że z nazw tych można korzystać w dowolnym miejscu wyrażenia ścieżkowego. Przestrzenie nazw dostępne są jako katalogi w katalogu głównym wirtualnego systemu plików. W zależności od wybranej przestrzeni nazw semantyka pewnych wywołań systemowych może się różnić.

Type alias *T*
: Przestrzeń nazw dla typów. Zdefiniowane są w niej typy wbudowane, a także funkcje wykorzystywane podczas konstrukcji typów zdefiniowanych przez użytkownika, deklaracji typów atrybutów itp.

  - typy wbudowane (przykłady): `@Type.Integer`, `@Type.String`
  - typy zdefiniowane przez użytkownika (przykłady):
    - *@Type.User.Status*, typ wyliczeniowy z możliwymi wartościami:
      * *draft*
      * *reviewed*
      * *approved*
    - *@Type.User.Id3Tag*, typ strukturalny z polami nazwanymi:
      * *title[@Type.String]*
      * *author[@Type.String]*
      * *date[@Type.Date]*
      * ...

  Nazwy typów muszą zaczynać się od wielkiej litery.

Tag
: Przestrzeń nazw dla etykiet. W tej przestrzeni nazw możliwe jest zarządzanie hierarchią etykiet. W przestrzeni zdefiniowane są również funkcje pomocnicze do przeglądania i zmiany poddrzew etykiet.

  - `@Tag.all` (??, *str.*??)

Attribute alias *Attr*
: Przestrzeń nazw w której możliwe jest zarządzanie atrybutami. Zdefiniowane atrybutu polega na określeniu typu jego wartości. Typy z których można korzystać są zdefiniowane w przestrzeni nazw *Type*.

Function aliasy: *F*, *Func*
: Przestrzeń nazw dla różnego rodzaju funkcji. Mogą się tu znaleźć zarówno funkcje, które jako argument przyjmują zapytanie, ale też funkcje dla których argumentami są wyłącznie typy wbudowane (*Integer*, *Real*, *String*, *Time*, *Date*). (**rozbudować**)

  - `@F.limit[10]` ??
  - `@F.limit[10].from[20]` ??
  - `@F.order` ??
  - `@F.order[@F.desc].by[@name]` ??
  - `@F.order.by[@name].using[@F.strcmp]` ??

User alias *U*
: Podprzestrzeń nazw dla innych przestrzeni nazw np.: *Type*, *Function*. (**czy na pewno potrzebna?**)

4.2.3 Dodawanie plików do indeksu

Przedstawiony jest najprostszy sposób dodawania plików do indeksu. W ramach poniższego przypadku użycia wykorzystana jest także możliwość listowania zawartości katalogu ale bez jakiegokolwiek filtrowania wyników.

Warunki początkowe

- w systemie nie ma jeszcze referencji do żadnego pliku

Scenariusze pozytywne

```
1 $ cd /vhfs/@File
2 $ ln -s /home/user/project/vhfs/README ./
3 $ ln -s /home/user/project/linux/README ./
4 $ ln -s /home/user/MichalUrbaniak.mp3 /vhfs/@File
5 $ ln -s /home/user/project/vhfs/README ./
6 $ ls -l /vhfs/@File
7 ... README.1 -> /home/user/project/vhfs/README
8 ... README.2 -> /home/user/project/linux/README
9 ... MichalUrbaniak.3.mp3 -> /home/user/MichalUrbaniak.mp3
```

- **linia 1** - Przejście do przestrzeni nazw *File*
- **linia 2** - Tworzone są referencje do pliku z domyślnym aliasem (identycznym jak nazwa pliku - jest to cecha narzędzia systemowego *ln*, które tworzy link o tej samej nazwie w momencie, gdy nie podamy nazwy docelowej).
- **linia 3** - Dodany jest plik o identycznej nazwie do wcześniej dodanego.
- **linia 4** - Do indeksu dodany jest trzeci plik, tym razem zawierający roszerzenie nazwy (w tym przypadku mp3).
- **linia 5** - Próba utworzenia referencji do pliku, który już został dodany do indeksu nie zgłasza błędu, ale powoduje, że stan systemu plików nie ulega zmianie.
- **linia 6** - odczytano całą zawartość indeksu (wynik zaprezentowany został w formie skrótowej, ponieważ pierwsze kolumny wyniku nie są nośnikiem ważnej z punktu widzenia tego przypadku informacji)

Scenariusze negatywne

```
1 $ ln -s /home/user/cv.pdf /vhfs/employee_cv.pdf
```

- **linia 1** - Dla uproszczenia zakładamy, że alias do pliku musi być identyczny jak nazwa pliku w systemie, a więc próba utworzenia pliku o innym aliasie w wirtualnym systemie plików kończy się niepowodzeniem. Docelowo powinno być umożliwione nadawanie aliasów.

Warunki końcowe

- Do indeksu dodane zostały trzy referencje do różnych plików.
- Od tej chwili próba listowania plików z systemu plików zwróci wynik niepusty. Każda próba filtrowania spowodowałaby usunięcie wszystkich referencji plików z wyniku zapytania ścieżkowego, ponieważ plikom nie nadano etykiet ani atrybutów.

Uwaga

Wartym odnotowania są zmodyfikowane nazwy plików. Ze względu na to, że indeksowanych może być wiele plików, które mają tą samą nazwę poza wirtualnym systemem plików (np. wiele plików o nazwie *README*, co w tradycyjnym systemie plików nie stanowi problemu tak długo, jak pliki te znajdują się w innych katalogach) konieczne jest rozróżnienie tych plików w wirtualnym katalogu. Wybraną metodą jest rozdzielenie nazwy pliku na bazową część nazwy (ang. *basename*) i część rozszerzenia (ang. *extension*) (część nazwy pliku od ostatniej kropki

do końca). Następnie tworzymy nowy alias dla pliku według schematu: "*$(basename).$(id).$(extension)*", gdzie *id* jest unikalnym numerycznym identyfikatorem pliku.

4.2.4 Tworzenie etykiet pierwszego rzędu

Każda etykieta, bez względu na miejsce w drzewie etykiet (*4.2.1, str.21*) w chwili utworzenia staje się obiektem globalnym, do którego dostęp możemy uzyskać poprzez jej nazwę poprzedzoną *@*. Pojęcie obiektu globalnego odpowiada pojęciu zmiennych o dostępie globalnym w imperatywnych językach programowania takich jak *C*, *Pascal* czy *Java*. Oznacza to, że nie mogą istnieć dwie etykiety pierwszego rzędu o tej samej nazwie. Przedstawione scenariusze dotyczą wymagań względem sposobu tworzenia etykiety pierwszego rzędu. (*4.2.1, str.21*).

Warunki początkowe

- w systemie nie zostały utworzone etykiety oraz referencje do plików

Scenariusze pozytywne

```
1 $ cd /vhfs/@Tag
2 $ mkdir @music
3 $ mkdir @doc
4 $ ls /vhfs/@File
5 $ ls /vhfs/@File/@music/
6 $ ls /vhfs/@File/@music/@doc
7 $ ls /vhfs/@Tag
8   @music   @doc
```

- **linia 1** - Zarządzanie etykietami odbywa się w przestrzeni nazw *Tag*, dla skróconego zapisu reszty wywołań przechodzimy do katalogu skojarzonego z tą przestrzenią.
- **linia 2** - Tworzy etykietę pierwszego rzędu o nazwie *music*.
- **linia 3** - Tworzy etykietę pierwszego rzędu o nazwie *doc*.
- **linia 4** - W systemie plików nie dodano do indeksu jeszcze ani jednego pliku, więc wynik wywołania narzędzia systemowego *ls* musi zwrócić pustą listę plików
- **linia 5** - Z tego samego powodu co powyżej, nie istnieją pliki, które mają przypisaną etykietę *music*
- **linia 6** - Z tego samego powodu co w dwóch poprzednich przypadkach nie istnieją pliki, które posiadają etykiety *music* i *doc* jednocześnie
- **linia 7** - Listowane są wszystkie etykiety, które są korzeniem jakiegoś poddrzewa etykiet.

Scenariusze negatywne

Załóżmy, że warunki początkowe zostały zmienione przez scenariusz pozytywny.

```
1 $ ls /vhfs/@File/@document
2 ls: cannot access /vhfs/@document:
3     No such file or directory
4
5 $ mkdir /vhfs/@Tag/@music
6   mkdir: cannot create directory /vhfs/@Tag/@music:
7         File exists
8 $ mkdir /vhfs/@File/@music
9   mkdir: cannot create directory /vhfs/@File/@music:
10        File exists
```

- **linia 1** - Odczyt katalogu zakończony niepowodzeniem, ponieważ nie istnieje w systemie etykieta *document*, a więc katalog taki również nie istnieje
- **linia 5** - Powtórna próba utworzenia etykiety *music* powinna zakończyć się niepowodzeniem.
- **linia 8** - Nie powinna powieść się też próba utworzenia jakielkowiek etykiety w trybie *File*.

Warunki końcowe

- Powstały etykiety pierwszego rzędu (*music*, *doc*).
- Żadna z nowo powstałych etykiet nie została przypisana do pliku.
- Nie dodano do indeksu żadnego pliku.

Uwaga

- każda etykieta (a także każdy inny identyfikator globalny) poprzedzony jest prefiksem `@`, czego przyczyna leży w interfejsie systemu plików, czego szczegóły czytelnik może poznać przechodząc do projektu implementacji dla operacji **getattr** (**??**, *str.***??**).

4.2.5 Nadawanie plikom etykiet i filtrowanie po etykiecie

Przypadki użycia dotyczą możliwości przypisywania etykiet do plików. Celem takiego przypisania jest możliwość filtrowania zbioru plików po etykiecie. Wymaganie to może ulec zmianie w późniejszym czasie, ponieważ założenie, że etykieta musi być utworzona w systemie przed przypisaniem jej do pliku nie jest jedynym możliwym podejściem. Alternatywą byłoby tworzenie etykiet z momencie tworzenia asocjacji pomiędzy etykietą a plikiem. Wówczas tworzenie etykiety byłoby możliwe za pomocą co najmniej dwóch

mechanizmów: tworzenia etykiety w opisany wcześniej sposób oraz poprzez przypisanie identyfikatora do pliku z jednoczesnym założeniem, że skoro identyfikator nie jest powiązany z żadnym istniejącym obiektem, to jest on przyszłą etykietą. Drugie podejście wiąże się jednak ze skomplikowaniem systemu oraz nie jest jednoznacznie korzystne z punktu widzenia użytkowania systemu. Z wymienionych powodów nie rozmważamy w tej chwili takiego podejścia.

Warunki początkowe

- w systemie utworzone zostały etykiety pierwszego rzędu: *music* oraz *movie*
- referencja do żadnego pliku nie została jeszcze dodana do systemu plików

Scenariusze pozytywne

```
1  $ cd /vhfs/@File
2  $ ln -s /files/Inception_Trailer.mpg @movie
3  $ ln -s /home/user/files/music.ogg @music
4  $ ln -s /files/ShindlerList1.mp3 @music/@movie
5  $ ls -l
6     ... Inception_Trailer.1.mpg -> /files/Inception_Trailer.mpg
7     ... music.2.ogg -> /home/user/files/music.ogg
8     ... ShindlerList1.3.mp3 -> /files/ShindlerList1.mp3
9  $ ls -l @movie
10    ... Inception_Trailer.1.mpg -> /files/Inception_Trailer.mpg
11    ... ShindlerList1.3.mp3 -> /files/ShindlerList1.mp3
12 $ ls -l @music
13    ... music.2.ogg -> /home/user/files/music.ogg
14    ... ShindlerList1.3.mp3 -> /files/ShindlerList1.mp3
15 $ ls -l @music/@movie
16    ... ShindlerList1.3.mp3 -> /files/ShindlerList1.mp3
```

- **linia 2** - Plikowi nadana została etykieta *movie*.
- **linia 3** - Plikowi nadana została etykieta *music*.
- **linia 4** - Plikowi nadane zostały jednocześnie dwie etykiety *music* oraz *movie*.
- **linia 5** - Wylistowane zostały wszystkie pliki w indeksie bez żadnego filtrowania.
- **linia 9** - Wylistowane zostały wszystkie pliki, które posiadają etykietę *movie*.
- **linia 12** - Wylistowane zostały wszystkie pliki, które posiadają etykietę *music*.
- **linia 15** - Wylistowane zostały wszystkie pliki, które posiadają jednocześnie etykietę *music* oraz *movie*.

Scenariusze negatywne

Załóżmy, że stan początkowy został zmieniony przez powyższe scenariusze pozytywne.

```
1 $ ln -s /files/music.mp3 /vhfs/@File/@jazz
2     ln: creating symbolic link '/vhfs/@jazz/music.mp3':
3         No such file or directory
```

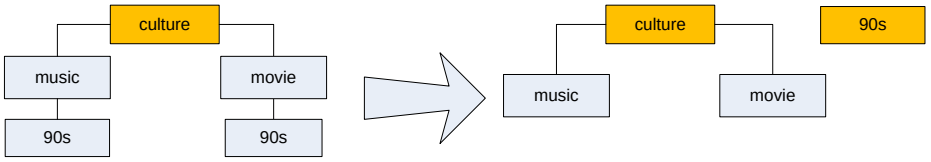
- **linia 1** - System powinien zwrócić błąd jeśli próbuje się plikowi nadać etykietę, która nie istnieje.

Warunki końcowe

- stworzono dwie nowe etykiety: *music*, *movie*
- do indeksu dodane zostały trzy pliki
- wszystkie dodane pliki posiadają etykiety, jeden z plików posiada dwie etykiety

4.2.6 Tworzenie podetykiet

W związku z tym, że każda nazwa etykiety jest jej identyfikatorem globalnym, to również nazwy podetykiet mają tę właściwość. Konsekwencją jest niemożność występowania etykiety o tej samej nazwie w dwóch poddrzewach etykiet. To pozorne organiczenie ma jednak swoje uzasadnienie - zniechęca do tworzenia podetykiet dla plików, jeśli pojęcie do którego etykieta się odwołuje jest ogólne i nie określa tylko podzbioru plików skojarzonych z pojęciem zdefiniowanym wyłącznie przez jedną nadetykietę. Inaczej mówiąc podetykiety są odpowiednie jeśli pojęcie definiowane przez natetykietę może być jednoznacznie zawężone. Matematycznie podetykietę można traktować jako klasę abstrakcji dla swojej nadetykiety. (**potwierdzić + przypis**)



Rysunek 4.1: Organizacja drzewa etykiet

W załączonym diagramie widzimy dwa rysunki odzwierciedlające drzewiastą strukturę etykiet. Etykiety pierwszego rzędu są w nim oznaczone na kolor złoty. Na diagramie po lewej względem symbolu strzałki widzimy podwójne użycie tej samej etykiety *90s* jako podetykiety (etykieta ta miała służyć do skojarzenia plików z pojęciem lat dziewięćdziesiątych). Taki sposób organizacji drzewa etykiet nie powinien być dopuszczony przez system,

ponieważ nie powinien on zezwolić na dwukrotne utworzenie podetykiety o tej samej nazwie. Po prawej stronie widzimy poprawioną strukturę, która powinna być dopuszczona przez system, w której kłopotliwa podetykieta *90s* stała się etykietą pierwszego rzędu bez podetykiet.

Najważniejsze w koncepcie podetykiet jest dziedziczenie nadetykiety, które następuje w momencie skojarzenia pliku z określoną podetykietą. Jeśli plik można skojarzyć z podetykietą pewnej etykiety, to skojarzenie danego pliku z tą detykietą powinno również mieć sens. Jeśli pojęcie klasyfikuje zbiory plików charakteryzowanych przez dwa inne pojęcia (podetykieta "pasuje" do dwóch nadetykiet), to jest to zdaniem autora uzasadniona motywacja aby traktować je przynajmniej na równi z tymi dwoma z punktu widzenia umiejscowienia na drzewie etykiet.

Dla każdego użytkownika relacja rodzic-dziecko może być inna. Każdy użytkownik systemu ma możliwość dowolnej organizacji drzewa etykiet w sposób, który odpowiada jego potrzebom. Organizacja drzewa etykiet może być różnorodna jak języki, którymi posługują się różne grupy zawodowe. Dla muzyków etykieta *csharp* może być pojęciowo czymś zupełnie różnym niż dla programistów. Ci pierwsi prędzej umieszczą *csharp* w poddrzewie etykiety *music*, natomiast ci drudzy odpowiednie miejsce dla niej mogą odnaleźć w poddrzewie etykiety *programming* jako podetykietę *language*. Jeszcze inaczej strukturę drzewiastą etykiet mógłby chcieć zorganizować programista zainteresowany nauką języków naturalnych - on uczyniłby etykietę *language* etykietą pierwszego rzędu, natomiast *csharp* mogłoby być podetykietą *programming*. Wydaje się więc, że nadmierne stosowanie mechanizmu podetykiet nie jest praktyką dobrą, bo mało uniwersalną. Zawężamy pojęcia skojarzone z etykietami do podklas pojęć reprezentowanych przez nadetykiety, co nie jest dobre w przypadku pojęć, które są znaczeniowo ortogonalne. Niemniej jednak w pewnych okolicznościach tworzenie podetykiet może być przydatne, dlatego wymaganie takie uznane zostało za istotne dla projektowanego systemu.

Warunki początkowe

- Utworzone zostały etykiety *music* oraz *movie*

Scenariusze pozytywne

```
$ ls /vhfs/@File/@music
$ cd /vhfs/@Tag
$ ls @music
$ mkdir @music/@jazz
$ ls /vhfs/@File/@music
$ ls @music
  @jazz
$ cd @music
$ mkdir @rap
$ ls
  @jazz @rap
```

- **linia 1** - Nie przyporządkowano plików do etykiety *music*, więc wynik wywołania *ls* zwraca pusty wynik.
- **linia 3** - Etykieta *music* nie posiada podetykiet, więc wynik wywołania *ls* zwraca pusty wynik.
- **linia 4** - Utworzona zostaje podetykieta *jazz* dla której nadetykietą jest *music*.
- **linia 5** - Nie istnieją pliki, które mają przyporządkowanie do etykiety *music*, więc *ls* nie zwraca listy plików.
- **linia 6** - Wynik wskazuje na utworzenie relacji rodzic-dziecko pomiędzy *music* a *jazz*.
- **linia 9** - Utworzona zostaje jeszcze jedna podetykieta *rap*. Jej nadetykietą jest *music*.
- **linia 10** - Listowanie wszystkich dzieci *music* w momencie gdy katalog roboczy został zmieniony i odpowiada miejscu zdefiniowania danej etykiety.

Scenariusze negatywne

```
1 $ cd /vhfs/@Tag
2 $ ls
3   @music @movie
4 $ mkdir @music/@movie
5   Operation Not supported
6 $ mkdir @music/@T.Integer
7   Operation Not supported
8 $ mkdir @music.children/@ROCK
9   Operation Not supported
```

- **linia 4** - System nie powinien zezwolić na utworzenie etykiety o nazwie takiej jak nazwa istniejącej etykiety.
- **linia 6** - Ostatni element wyrażenia ścieżkowego nie jest poprawnym identyfikatorem etykiety (jest typem), więc niemożliwe jest utworzenie etykiety.
- **linia 8** - *ROCK* również nie jest poprawnym identyfikatorem dla etykiety dlatego system nie powinien utworzyć etykiety o takiej nazwie.

Warunki końcowe

- Zostały utworzone podetykiety: *rock*, *rap* dla których nadetykietą jest: *music*

4.2.7 Usuwanie etykiety

Usunięcie etykiety wiąże się ze zniszczeniem asocjacji pomiędzy plikami a daną etykietą. Dla uproszczenia projektu systemu nie wymaga się możliwości automatycznego tworzenia asocjacji pomiędzy plikami a nadetykietą usuwanej podetykiety. Tym bardziej nie wymaga się aby typ takiego zachowania był konfigurowalny.

Warunki początkowe

- W systemie utworzono etykietę: *music*
- W systemie utworzono podetykiety *jazz* oraz *rock* dla których nadetykietą jest *music*
- Etykietę *music* przypisano dwóm plikom:
  - `/f/file.mp3`
  - `/f/file2.mp3`
- Podetykietę *jazz* przypisano dwóm plikom:
  - `/f/file.mp3`
  - `/f/jazz.mp3`
- Podetykiety *rock* nie przypisano żadnym plikom.

Scenariusze pozytywne

```
1 $ cd /vhfs/@File
2 $ ls -l @music
3 ... file.1.mp3 -> /f/file.mp3
4 ... file2.2.mp3 -> /f/file2.mp3
5 ... jazz.3.mp3 -> /f/jazz.mp3
6 $ ls -l @jazz
7 ... file.1.mp3 -> /f/file.mp3
8 ... jazz.3.mp3 -> /f/jazz.mp3
9 $ rmdir /vhfs/@Tag/@music/@jazz
10 $ ls /vhfs/@Tag/@music
11   @rock
12 $ ls @music
13 ... file.1.mp3 -> /f/file.mp3
14 ... file2.2.mp3 -> /f/file2.mp3
15 $ rm -rf /vhfs/@Tag/@music
16 $ ls /vhfs/@Tag
17 $
```

- **linia 2** - Drukuje wszystkie pliki z etykietą *music*. Wydruk zawiera także wszystkie pliki, które posiadają asocjację z etykietą *jazz*.
- **linia 6** - Drukuje wszystkie pliki z etykietą *jazz*.
- **linia 9** - Usuwa etykietę *jazz*, które odbywa się w dwóch etapach:

  - Usuwa etykietę *jazz* ze wszystkich plików.
  - Usuwa etykietę *jazz* tak, że nie będzie można jej już przypisać do żadnego pliku, bez ponownego utworzenia z wykorzystaniem wywołania systemowego *mkdir*

- **linia 10** - Sprawdza, czy zgodnie z założeniem usunięta została podetykieta *jazz*.
- **linia 12** - Z wydruku zniknął plik, który posiadał asocjację z nieistniejącą już podetykietą *jazz* dla której nadetykietą jest *jazz*.
- **linia 15** - Usuwa etykietę *music* wraz z etykietami (w tej chwili tylko *rock*).
- **linia 16** - Drukuje listę wszystkich etykiet. Razem z etykietą *music* usunięta została także etykieta *rock*, a zatem w systemie nie pozostała już żadna etykieta.

Scenariusze negatywne

(**Po zmianach z przestrzeniami nazw przykład i opis są nieadekwatne.**) Załóżmy, że system plików jest w stanie początkowym (nie zostały dodane żadne pliki, etykiety, atrybuty).

```
1  $ cd /vhfs
2  $ mkdir @Tag/@music
3  $ mkdir @Tag/@movie
4  $ ln -s /f/JohnWilliams.mp3 @File/@music/@movie
5  $ ln -s /f/MilesDavis.mp3 @File/@music
6  $ ls -l @File/@music/@movie
7    ... JohnWilliams.1.mp3 -> /f/JohnWilliams.mp3
8  $ ls -l @File/@music
9    ... JohnWilliams.1.mp3 -> /f/JohnWilliams.mp3
10   ... MilesDavis.2.mp3 -> /f/MilesDavis.mp3
11 $ rm -rf @File/@music/@movie
12   rm: cannot remove /vhfs/@File/@music/@movie:
13       operation not supported
14 $ ls -l @music
15   ... MilesDavis.2.mp3 -> /f/MilesDavis.mp3
16 $ ls -l @movie
17 $
```

- **linia 11** - Usunięcie etykiet *music* oraz *movie* ze wszystkich plików, które posiadają przyporządkowanie do tych etykiet. Nie następuje usunięcie samej etykiety *movie*, ani tym bardziej etykiety *music*. Ponieważ usuwanie etykiet w trybie pracy z przestrzenią *File* nie jest możliwe.
- **linia 14** - Wyświetlenie wyników wyszukiwania referencji do plików zawierających asocjację z *music*.
- **linia 16** - Wyświetlenie wyników wyszukiwania referencji do plików zawierających asocjację z *movie*. Dostęp do katalogu

`/vhfs/@File/@movie` jest ciągle możliwy ponieważ nie została usunięta etykieta *movie*.

Warunki końcowe

- W przypadku scenariuszy pozytywnych w systemie nie pozostały żadne etykiety, ani referencje do plików.
- W przypadku scenariuszy negatywnych obydwie etykiety nadal istnieją w systemie.

4.2.8 Usuwanie plików z indeksu

Wymaganiem równie naturalnym jak możliwość dodawania plików do indeksu jest możliwość ich usuwania.

Warunki początkowe

- W systemie utworzono etykiety *music* oraz *movie*
- W systemie zadeklarowano atrybut *title* z typem wartości *String*

Scenariusze pozytywne

```
1 $ cd /vhfs/@File
2 $ ln -s /f/ShindlerList.mp3 @music/@movie/@title="{Shindler List}"
3 $ ls -l
4   ... ShindlerList.1.mp3 -> /f/ShindlerList1.mp3
5 $ rm ShindlerList1.1.mp3
6 $ ls -l
7 $
```

- **linia 2** - Dodanie pliku do indeksu wraz z utworzeniem asocjacji a etykietami oraz nadaniem atrybutu.
- **linia 3** - Sprawdzenie, czy plik został dodany do indeksu.
- **linia 5** - Usunięcie pliku z indeksu.
- **linia 6** - Sprawdzenie, czy plik został usunięty.

Scenariusze negatywne

```
1 $ cd /vhfs/@File
2 $ ln -s /f/ShindlerList.mp3 @music/@movie/@title="{Shindler List}"
3 $ ls -l
4  ... ShindlerList.1.mp3 -> /f/ShindlerList1.mp3
5 $ rm @music/@movie/@title="{Shindler List}"/ShindlerList1.1.mp3
6 $ ls -l  @music/@movie/@title="{Shindler List}"
7 $ ls -l
8  ... ShindlerList.1.mp3 -> /f/ShindlerList1.mp3
```

- **linia 5** - Usunięcie wszystkich stworzonych uprzednio asocjacji pliku z atrybutami oraz etykietami.

- **linia 6** - Sprawdzenie czy usunięte zostały w/w asocjacje.
- **linia 7** - Plik jest nadal w indeksie. Jedyny sposób na usunięcie pliku to usunięcie go bez żadnych filtrów (warunków na atrybutach i etykietach).

Warunki końcowe

- W przypadku scenariuszy pozytywnych wyjściem jest usunięcie pliku z indeksu.
- W przypadku scenariuszy negatywnych wyjściem jest pozostanie pliku w indeksie.

4.2.9 Deklaracja atrybutów

Aby zacząć używać nowego atrybutu, tj. nadawać atrybuty plikom, filtrować pliki z wykorzystaniem atrybutu, należy zdefiniować dziedzinę jego wartości. Podane niżej przypadki zakładają definicję atrybutów dla których dziedzina wartości określona jest przez typ wbudowany, ale dla typów zdefiniowanych przez użytkownika procedura powinna wyglądać analogicznie, z tym, że odpowiedni typ musi być już zdefiniowany w chwili deklarowania atrybutu.

Warunki początkowe

- Nie zostały zadeklarowane żadne atrybuty.

Scenariusze pozytywne

```
1 $ cd /vhfs/@Attribute
2 $ ls
3 $ mkdir @title=@Type.String
4 $ mkdir @year=@Type.Integer
5 $ ls /vhfs/@File/@title="{Inception}"
6 $ ln -s /f/Inception.avi /vhfs/@File/@title="{Inception}"/@year=2010
7 $ ls /vhfs/@File/@title="{Inception}"/@year=2010
8   ... Inception.1.avi -> /f/Inception.avi
```

- **linia 1** - Przejście do przestrzeni *Attribute* w której możliwe jest zarządzanie atrybutami.
- **linia 3** - Zadeklarowanie atrybutu *title* o typie wbudowanym *String*.
- **linia 4** - Zadeklarowanie atrybutu *year* o typie wbudowanym *Integer*.
- **linia 5** - Filtrowanie plików po tytule (typ przyrównywanej wartości jest zgodny z zadeklarowanym dla atrybutu *title*, a więc system nie zgłasza błędu - np. nieistnienie katalogu).

- **linia 7** - Filtrowanie plików za pomocą warunków określonych na dwóch atrybutach.

Scenariusze negatywne Załóżmy, że stan systemu pilków zmienił się na skutek wykonania scenariuszy pozytywnych.

```
1 $ cd /vhfs/@File
2 $ ls @title=12
3   ls: cannot access @title=12: No such file or directory
4 $ ln -s /f/music.mp3 @title=12
5   ln: creating symbolic link '@title=12': No such file or directory
6 $ ls @year="{2000}"
7   ls: cannot access @year="{2000}": No such file or directory
```

- **linia 2** - Warunek filtrowania jest niezgodny z zadeklarowanym typem dla atrybutu *title*, dlatego system informuje, że nie istnieje taki katalog.
- **linia 4** - Próba przypisania wartości dla atrybutu spoza dziedziny atrybutu.
- **linia 6** - Próba przypisania wartości do atrybutu spoza jego dziedziny - w przyszłości taki typ przypisania może być obsługiwany poprzez niejawne rzutowanie.

4.2.10 Nadawanie plikom atrybutów

Dla uproszczenia pierwszej wersji systemu zakładamy, że system powinien wspierać tylko atrybuty o kardynalnościach 0 i 1. Oznacza to, że każdy plik może posiadać lub nie zadeklarowany w systemie atrybut z wartością o określonej dziedzinie, nie może jednak posiadać wielu atrybutów o tej samej nazwie.

Warunki początkowe

- Referencja do żadnego pliku nie została jeszcze dodana do systemu plików.
- Zadeklarowano atrybuty:
  - *year*, którego wartości są wbudowanego typu *Integer*
  - *title*, którego wartości są wbudowanego typu *String*

Scenariusze pozytywne

```
1 $ cd /vhfs/@File
2 $ ln -s /f/music.mp3 @year=1985
3 $ ln -s /f/music.mp3 @title="{It's my life}"
4 $ ln -s /f/urbaniak.mp3 @year=1985/@title={Free}
5 $ ls -l @year=1985
6    ... music.1.mp3 -> /f/music.mp3
```

```
7     ... urbaniak.2.mp3 -> /f/urbaniak.mp3
8  $ ls -l @title={Free}
9     ... urbaniak.2.mp3 -> /f/urbaniak.mp3
```

- **linia 2** - do indeksu został dodany plik i ustawiono atrybut *year* z wartością *1985* typu *Integer*
- **linia 3** - do indeksu został dodany plik i ustawiono mu atrybut *title* z wartością *"It's my life"* typu *String*
- **linia 5** - wylistowane zostały pliki, które posiadają atrybut *year* z ustawioną wartością *1985*
- **linia 8** - wylistowane zostały pliki, które posiadają atrybut *title* z ustawioną wartością *"Free"*

Scenariusze negatywne Załóżmy, że stan początkowy zmienił się pod wpływem wykonania powyższych scenariuszy pozytywnych.

```
1  $ cd  /vhfs/@File
2  $ ln -s /f/doc/agreement.pdf @author="{Pigs and eggs Company}"
3     blad!
4  $ ln -s /f/music/urbaniak.mp3 @year=1995.5
5     błąd!
```

- próba nadania atrybutu, który nie został uprzednio zadeklarowany kończy się niepowodzeniem
- próba przypisania wartości spoza dziedziny atrybutu kończy się niepowodzeniem (wartość jest typu *Real*, natomiast deklaracja atrybutu zakładała jego dziedzinę typu *Integer*)

Warunki końcowe

- zadeklarowane zostały typy atrybutów dla *year* oraz *title*
- nadano atrybuty *year* oraz *title* z wartościami odpowiadających im dziedzin

Uwaga

- Używamy specjalnych ciągów znaków (szerzej o tym w gramatyce języka 4.5.4).
- Jako ogranicznik dla literału ciągu znaków (ang. *String*) używamy nawiasów klamrowych, stosowanie dodatkowych cudzysłowów jest konieczne jeśli w ciągu znaków występują znaki białe, ponieważ bez tego powłoka potraktuje ciąg jako listę parametrów.
- Znaki cudzysłowu są wymuszone poprzez powłokę, nie są częścią języka.

4.2.11 Usuwanie etykiet z plików

Przedstawione przypadki użycia prezentują sposób usuwania etykiet z plików, bez usuwania etykiet z systemu. W odróżnieniu od usuwania etykiet z systemu w przypadku usuwania etykiet z plików wszystkie asocjacje etykiety z innymi plikami pozostają niezmienione.

Warunki początkowe

- utworzone zostały etykiety *music* oraz *movie*

Scenariusze pozytywne

```
1  $ cd /vhfs/@Tag
2  $ ls @music
3  $ ls @movie
4  $ cd /vhfs/@File
5  $ ln -s /f/JawsSoundtrack.mp3 @music/@movie
6  $ ls -l /vhfs
7    ... JawsSountrack.1.mp3 -> /f/JawsSountrack.mp3
8  $ ls JawsSoundtrack.1.mp3.tags
9    @movie @music
10 $ rm @movie/JawsSoundtrack.1.mp3
11 $ ls @movie
12 $ ls -l @music
13   ... JawsSountrack.1.mp3 -> /f/JawsSountrack.mp3
14 $ rm JawsSountrack.1.mp3.tags/@music
15 $ ls @music
16 $
```

- **linia 2** - listuje wszystkie pliki z etykietą *music*
- **linia 3** - listuje wszystkie pliki z etykietą *movie*
- **linia 5** - nadaje etykiety *music* oraz *movie* plikowi, jednocześnie dodając go do indeksu
- **linia 8** - listuje wszystkie etykiety dla danego pliku (metoda pomocnicza na plikach)
- **linia 10** - usuwa etykietę z danego pliku (pierwszy sposób)
- **linia 11** - sprawdza czy rzeczywiście etykieta *movie* została usunięta z pliku
- **linia 12** - sprawdza, czy przypisanie pozostałych etykiet się nie zmieniło
- **linia 14** - usuwa etykietę *music* z pliku (drugi sposób)
- **linia 15** - sprawdza, czy etykieta rzeczywiście została usunięta

Scenariusze negatywne

```
1 $ ln -s /f/JohnWilliams.doc @movie/@music
2 $ ls -l @movie/@music
3  ... JohnWilliams.2.doc -> /f/JohnWilliams.doc
4 $ rm @movie/JohnWilliams.2.doc
5 $ ls -l @movie
6 $ rm @movie/JohnWilliams.2.doc
7  rm: cannot remove '@movie/JohnWilliams.2.doc': No such file or directory
```

- **linia 4** - Usunięcie etykiety *movie* z pliku.
- **linia 5** - Sprawdzenie, czy etykieta została faktycznie usunięta.
- **linia 6** - Ponowna próba usunięcia powinna się nie powieść, ponieważ plik mimo, że istnieje w indeksie, to nie istnieje asocjacja tego pliku z etykietą *movie*.

Warunki końcowe

- Żaden z plików nie ma zachowanej asocjacji z jakąkolwiek etykietą.
- Referencje do plików nadal istnieją w systemie.

4.2.12 Usuwanie atrybutów z plików

Usuwanie atrybutów z plików jest operacją podobno do usuwania etykiet z plików.

Warunki początkowe

- zadeklarowane zostały atrybuty:
  - *bitrate* z typem *Integer*
  - *title* z typem *String*
- nie dodano żadnych plików do indeksu

Scenariusze pozytywne

```
1 $ cd /vhfs/@File
2 $ ln -s /f/JohnWilliams.mp3 @bitrate=128/@title="{Indiana Jones theme}"
3 $ ls -l @bitrate=128
4 ...  /f/JohnWilliams.1.mp3 -> /f/JohnWilliams.mp3
5 $ ls -l @title="{Indiana Jones theme}"
6 ...  /f/JohnWilliams.1.mp3 -> /f/JohnWilliams.mp3
7 $ ls -l JohnwWilliams.1.mp3.attributes
8  @bitrate=128    @title="{Indiana Jones theme}"
9 $ rm @bitrate=128/JohnWilliams.1.mp3
10 $ ls @bitrate=128
11 $ ls JohnWilliams.1.mp3.attributes
12  @title="{Indiana Jones theme}"
13 $ rm JohnWilliams.1.mp3.attributes/@title="{Indiana Jones theme}"
14 $ ls @title="{Indiana Jones theme}"
15 $
```

- **linia 2** - Przypisanie plikowi dwóch atrybutów z odpowiednimi wartościami spełniającymi założenia o typach.
- **linia 9** - Usunięcie atrybutu *bitrate* z pliku.
- **linia 10** - Sprawdzenie, czy atrybut *bitrate* został usunięty z pliku.
- **linia 11** - Sprawdzenie wszystkich atrybutów pliku.
- **linia 13** - Usunięcie atrybutu *title* składna alternatywna. (**do powtórnego przemyślenia**)

Scenariusze negatywne Załóżmy, że warunki początkowe zostały zmienione przez scenariusze pozytywne.

```
1 $ rm @bitrate=128/JohnWilliams.1.mp3
2  błąd!
```

- **linia 1** - Próba usunięcie atrybutu, który nie jest przypisany do pliku powinna zakończyć się niepowodzeniem. (**dospecyfikować dokładnie sposób sygnalizacji o niepowodzeniu**)

4.2.13 Zmiana nazwy etykiety

System powinien umożliwiać zmianę nazwy etykiety. Zmiana nazwy etykiety nie powinna zmieniać asocjacji z plikami z którymi dana etykieta jest skojarzona. System nie powinien pozwalać na zmianę nazwy etykiety na nazwę, która jest identyfikatorem globalnym.

Warunki początkowe

- Zostały utworzone etykiety:
  - etykieta pierwszego rzędu: *music*
  - podetykieta *jazz*, dla której nadetykietą jest *music*:
- Został zadeklarowany atrybut *author* z typem wartości *String*

Scenariusze pozytywne

```
1 $ cd /vhfs/@Tag
2 $ ls
3   @music
4 $ mv @music @art
5 $ ls @art
6   @jazz
```

- **linia 4** - Etykieta *music* zmieniła nazwę na *art*.
- **linia 5** - Etykieta *art* jest teraz nadetykietą dla *jazz*, jedynego dziecka nieistniejącej już etykiety *music*

Scenariusze negatywne

```
1 $ cd /vhfs/@Tag
2 $ mv /vhfs/@art /vhfs/@author="George Orwell"
3   Operation not supported
4 $ mv /vhfs/@art /vhfs/@jazz
5   Operation not supported
```

- **linia 2** - System nie powinien zezwolić na zmianę nazwy etykiety na wyrażenie warunkowe na atrybucie. Docelowa nazwa musi spełniać warunki nazwy dla etykiet (*4.2.1, str.21*)
- **linia 4** - Etykieta *art* nie zmieniła nazwy. Nie została też usunięta etykieta *jazz*. System pozostał w niezmienionym stanie.

Warunki końcowe

- zmieniła się nazwa etykiety *music* na *art*

Uwaga

Relacja rodzic-dziecko jest zachowana po zmianie nazwy etykiety. Etykieta *music* po zmianie nazwy na *art* nadal posiada wszystkie swoje podetykiety.

4.2.14 Funkcje *limit* oraz *order by*

Odczyt zawartości katalogów to podstawowa funkcjonalność wirtualnego systemu plików. Do tej pory scenariusze użycia systemu plików pokazywały wykorzystanie tej funkcjonalności w ograniczonym zakresie. W niniejszym przypadku użycia zaprezentowane zostaną bardziej złożone metody odczytu zawartości katalogów, które powinny być wspierane przez system. Oprócz tradycyjnego wykorzystywania warunków filtrowania po atrybutach oraz etykietach wykorzystane zostaną

Warunki początkowe

- W systemie zostały zdefiniowane etykiety:
  - *music* z podetykietami:
    - *rock*
    - *rap*
    - *jazz* z podetykietami:
      - *smooth_jazz*
      - *nu_jazz*
      - *acid_jazz*
  - *movie* z podekietami:
    - *thriller*

        * *horror*
    - *favourite*
- W systemie zostały zdefiniowane też atrybuty:
    - *author* typu *String*
    - *year* typu *Integer*
- W systemie istnieją referencje do plików:
    - `/f/Urbaniak.mp3` z przypisanymi:
        * etykietami:
            · *smooth_jazz*
            · *favourite*
        * atrybutami:
            · *author* z wartością {*Michal Urbaniak*}
            · *year* z wartością *1985*
    - `/f/MilesDavis.mp3` z przypisanymi:
        * etykietami:
            · *music*
        * atrybutami:
            · *author* z wartością {*Miles Davis*}

Scenariusze pozytywne

```
$ cd /vhfs/@File
$ ls @music/@F.order[@F.@desc].by[@author]
 ... Urbaniak.1.mp3 -> /f/Urbaniak.mp3
 ... MilesDavis.2.mp3 -> /f/MilesDavis.mp3
$ cd @music/@F.order[@F.@asc].by[@author]
$ ls @F.limit[1]
 ... MilesDavis.2.mp3 -> /f/MilesDavis.mp3
$ ls @F.limit[1].from[1]
 ... Urbaniak.1.mp3 -> /f/Urbaniak.mp3
$ cd /vhfs/@File
$ ls @F.order.by[@year]
 ... Urbaniak.1.mp3 -> /f/Urbaniak.mp3
```

-

Scenariusze negatywne

```
        TODO
```

- (**wypełnić**)

4.2.15 Wsparcie dla atrybutów wbudowanych

Atrybuty wbudowane to takie, które są określone dla każdego pliku i wynikają z konstrukcji tradycyjnych systemów plików, do których wirtualny system plików przetrzymuje linki symboliczne. Każdy plik posiada pewien zbiór cech, takich jak:

- daty
  - modyfikacji
  - utworzenia
  - dostępu
- rozmiar pliku
- właściciela pliku
- grupę
- prawa dostępu dla właściciela, grupy, innych

Wszystkie te informacje mogą być wyłuskane, a potem na bieżąco aktualizowane przez system w przypadku zmian. Atrybuty te będąc gromadzone w wirtualnym systemie plików mogą posłużyć do filtrowania wyników wyszukiwania tak jak atrybuty nadane ręcznie.

Warunki początkowe

- Załóżmy, że wirtualny system plików znajduję się w stanie początkowym (czyli jest to pierwsze uruchomienie systemu plików, a stan danych nie został jeszcze zmieniony przez działalność użytkownika).

Scenariusze pozytywne

```
1 $ cd /vhfs/@File
2 $ ls -l /home/seba/
3 -rwxr-xr-x 1 seba users 4075 2010-08-15 20:42 null.py
4 $ stat /home/seba/null.py
5   File: 'null.py'
6   Size: 4074          Blocks: 8     IO Block: 4096 regular file
7 Device: 801h/2049d    Inode: 21082  Links: 1
8 Access: (0755/-rwxr-xr-x)  Uid: ( 1000/ seba) Gid: ( 105/ users)
9 Access: 2010-09-01 06:50:15.000000000 +0200
10 Modify: 2010-08-15 20:42:52.000000000 +0200
11 Change: 2010-08-15 20:42:52.000000000 +0200
12 $ ln -s /home/seba/null.py ./
13 $ ls -l
14  ... null.1.py -> /home/seba/null.py
15 $ ls -l @user={seba}/@group={users}/@size<5000
```

```
16  ... null.1.py -> /home/seba/null.py
17 $ ls -l ''@group in [{users},{adm},{root}]/@name like {null%}''
18  ... null.1.py -> /home/seba/null.py
```

- **linia 12** - Proste dodanie do indeksu referencji do istniejącego pliku.
- **linia 15** - Filtrowanie plików wyłącznie za pomocą warunków na atrybutach wbudowanych.
- **linia 17** - Filtrowanie plików wyłącznie za pomocą warunków wbudowanych z wykorzystaniem operatorów *like* oraz *in*.

Scenariusze negatywne

```
1 $ cd /vhfs/@File
2 $ ln -s /home/seba/NonExistantFile ./
3 ln: Operation not supported.
```

- (**wypełnić**)

4.2.16 Przesunięcie poddrzewa etykiet

Przesunięcie poddrzewa etykiet wiąże się ze zmianą bądź usunięciem nadetykiety dla etykiety, która jest korzeniem swojego poddrzewa. W przypadku usunięcia nadetykiety etykieta będąca korzeniem danego poddrzewa staje się etykietą pierwszego rzędu. Asocjacje z plikami, które posiadały skojarzenia z etykietami przenoszonego poddrzewa się nie zmieniają. Tym samym pliki, które posiadają takie skojarzenie mogą pojawić się w wynikach wyszukiwania gdy warunkiem wyszukiwania jest asocjacja z nową nadetykietą.

[DIAGRAM]

Warunki początkowe

- W systemie zostały zdefiniowane etykiety:
  - *music* z podetykietami:
    - *rock*
    - *jazz* z podetykietami:
      - *smooth_jazz*
      - *acid_jazz*
  - *movie* z podekietami:
    - *thriller*
    - *horror*
  - *favourite*

- Do indkesu dodany jest plik *MilesDavis.mp3* z przypisaniem do etykiet: *@nu_jazz* i *@smooth_jazz*.

Scenariusze pozytywne

```
1 $ cd /vhfs/@Tag
2 $ ls -l /vhfs/@File/@music
3  ... MilesDavis.1.mp3 -> /f/MilesDavis.mp3
4 $ ls -l /vhfs/@File/@favourite
5 $ mv @music/@jazz @favourite/@jazz
6 $ ls -l /vhfs/@File/@music
7 $ ls -l /vhfs/@File/@favourite
8  ... MilesDavis.1.mp3 -> /f/MilesDavis.mp3
9 $ ls /vhfs/@Tag/@favourite
10  @jazz
11 $ ls /vhfs/@Tag/@music/
12  @rock
```

- **linia 2** - Filtrowanie wszystkich plików, które zawierają asocjację z etykietą *music*.
- **linia 4** - Filtrowanie wszystkich plików, które zawierają asocjację z etykietą *favourite*.
- **linia 5** - Przeniesienie poddrzewa etykiet, którego korzeniem jest etykieta *jazz* z poddrzewa *music* do poddrzewa *favourite*.
- **linia 6** - Z plików posiadających skojarzenie z *music* (poprzez relację z podetykietą *jazz*) zniknęły te, które posiadały asocjację z *jazz*, która zmieniła nadetykietę.
- **linia 7** - Wyświetla pliki zawierające asocjację z etykietą *favourite*. Po zmianie nadetykiety dla *jazz* pliki posiadające asocjację z *jazz* lub jej podetykietami spełniają warunek filtrowania po nadetykiecie *favourite*.
- **linia 9** - Wyświetla zmienioną listę podetykiet *favourite*.
- **linia 11** - Wyświetla zmienioną listę podetykiet *music*.

Scenariusze negatywne

```
1 $ cd /vhfs
2 $ mv @Tag/@music @Tag/@music/@jazz/
3 mv: cannot move '@music' to a subdirectory of itself, '@music/@jazz'
4 $ mv @Tag/@music @File/@newmusic
5 mv: Operation not supported.
```

- **linia 2** - Narzędzie systemowe mv nie pozwoli wykonać tego typu operacji. Niemniej jednak system sam w sobie powinien zapewnić mechanizm obronny przed taką próbą.
- **linia 4** - System powinien zabronić próby przesunięcia katalogu z jednej przestrzeni nazw do innej.

4.2.17 Referencja wewnątrz wirtualnego systemu plików

```
1 ln -s /vhfs/music.123.mp3 /vhfs/Chopin.mp3
```

(**Do zrobienia**)

4.2.18 Typy zdefiniowane przez użytkownika

System powinien wspierać domyślnie następujące typy danych:

- *Real* dla liczb całkowitych
- *Integer* dla liczb całkowitych
- *String* dla ciągów znaków
- *Date* dla daty (dzień, miesiąc, rok)
- *Time* dla czasu (godzina, minuta, sekunda)

Proste typy danych są fundamentem dla złożonych typów danych. Przy czym nawet typy wbudowane takie jak: *Date* oraz *Time* są złożonymi typami danych, które są skonstruowane na bazie prostych typów danych (konkretnie *Integer*). W przyszłości powinno zostać umożliwione także tworzenie nowych typów danych. Jest to zagadnienie złożone, dlatego pierwsze wersje implementacji nie muszą zawierać tej funkcjonalności.

- *enum* dla obiektów typu wyliczeniowego
- *struct* dla obiektów typu złożonego

Warunki początkowe

- żadne typy atrybutów nie zostały jeszcze utworzone

Scenariusze pozytywne

```
1  $ cd /vhfs/@Type
2  $ ls
3   @Integer @String @Real @Date @Time
4  $ ln -s @Integer @Year
5  $ ls -l
6   d--- ... @Integer
7   d--- ... @String
8   d--- ... @Real
9   dr-x ... @Date
10  dr-x ... @Time
11  lrwx ... @Year -> ./@Integer
12 $ ls -l @Integer
13  ls: cannot open directory @Integer: Permission denied
14 $ cd @Date
15 $ ls -l
```

```
16 lrwx ... @day -> ../@Integer
17 lrwx ... @month -> ../@Integer
18 lrwx ... @year -> ../@Integer
19 $ mkdir @Id3Tag
20 $ cd @Id3Tag
21 $ ln -s /vhfs/@Type/@String @title
22 $ ln -s /vhfs/@Type/@String @author
23 $ ln -s /vhfs/@Type/@Date @date
24 $ ls -l
25 lrwx ... @title -> ../@String
26 lrwx ... @author -> ../@String
27 lrwx ... @date -> ../@Integer
28 $ cd ..
29 $ mkdir @Status
30 $ ln -s /vhfs/@Type.enum[0] @draft
31 $ ln -s /vhfs/@Type.enum @reviewed
32 $ ln -s /vhfs/@Type.enum @approved
33 $ ls @Status
34 lrwx ... @draft -> ../@Type.enum[0]
35 lrwx ... @reviewed -> ../@Type.enum[1]
36 lrwx ... @approved -> ../@Type.enum[2]
```

- (**wypełnić**)

Scenariusze negatywne

```
1 (wypełnić)
```

- (**wypełnić**)

4.2.19 Dynamiczne widoki

Dynamiczne widoki to mechanizm, który pozwala na utworzenie skrótu do warunku zdefiniowanego przez użytkownika. Celem skrótu może być dowolne wyrażenie ścieżkowe w systemie. Takie skrócone wyrażenie ścieżkowe można potem dowolnie rozszerzać tak jak każdą inną ścieżkę.

Warunki początkowe

-

Scenariusze pozytywne

```
1
2 $ cd /vhfs/
3 $ ln -s @File/@music/@jazz/@year<1990/''@author like Miles%'' @View/@the_best
```

- (**wypełnić**)

Scenariusze negatywne

```
1 (wypełnić)
```

- (**wypełnić**)

4.2.20 Dedukcja typu atrybutu

Być może niepotrzebne? (**Przemyśleć wady, zalety i pracę potrzebną do wykonania**)

```
$ cd /vhfs
$ mv /f/music.mp3 @genre="heavy metal"
```

4.2.21 Zmiana typu atrybutu

Byłaby to ewolucja schematu bazy danych pod spodem (raczej funkcjonalność na dalsze wydania).

4.3 Projekt rozwiązania

4.3.1 Operacje systemu plików

(**przedstawić sposób funkcjonowania podstawowych**)
Wszystkie poniższe operacje pochodzą z interfejsu systemu plików. Oferowane są pośrednio poprzez interfejs biblioteki fuse. (**rozbudować**)

4.3.2 Readdir

Operacja ta jest odpowiednikiem selekcji wykonanej na bazie danych. Możliwości operacji na wirtualnym systemie plików.

```

    /@music
    /@music/@jazz

    /@music/@bitrate < 128 # niejawne rzutowanie
    /@music/@bitrate < @t.kbps[128] // @t - przestrzen nazw dla typów, jawne rzutowanie

    /@bitrate.between[128, 192]
    /@bitrate.between[@t.kbps(128), @t.kbps(192)]

    /@music.children
    /@music.children[]

    /@music/@f.order
    /@music/@f.order.by(@music)
    /@music/@f.order[@desc].by[@id].using[@t.integer.lessThan]
    /@music/@f.limit[10]

    /@music/file__123.mp3.tags
    /@music/file_123.mp3.attributes

    /@name = file.mp3
```

```
    /@id = 123

    $ mkdir /vhfs/@music
    $ mkdir /vhfs/@music/@jazz
    $ mkdir /vhfs/@movie

    $ ls /vhfs/
    @music @movie
    $

    $ ls /vhfs/@music/
    $

    $ touch /vhfs/@music/file
    $ touch /vhfs/@jazz/file

    $ ls /vhfs/@music
    file_1 file_2
    $

    $ ls /vhfs/@jazz
    file_2
    $

```

- (**przykłady zaoytan**)

Mkdir

Operacja ta odpowiada operacji insert na bazie danych. Umożliwia tworzenie etykiet (ang. *tag*). (**rozbudować**)

Symlink

Operacja ta umożliwia nadawanie cech plikom - atrybuty oraz etykiety. Możliwe jest tworzenie widoków:

```
ln -s /home/user/documents/school/english/ernest_hemingway.doc
      /vhfs/@doc/@school/@learning/@homeworks/

Symlink(source=''/home/user/documents/school/english/ernest_hemingway.doc'',
        target=''@doc/@school/@learning/@homeworks'')


ln -s /vhfs/@documents/@job/order by modification/limit[10] /vhfs/@recent_docs

Symlink(source=''/vhfs/@documents/@job/order by modification/limit[10]'',
        target=''/vhfs/@recent_docs'')

```

(**rozbudować**)

Rename

Umożliwia usunięcie i dodanie etykiet oraz atrybutów do obiektów, zmiany nazw etykiet, oraz zmiany nazw atrybutów (?). (**rozbudować**)

Readlink

Operacja ta stanowi łącznik pomiędzy wirtualnym systemem plików a rzeczywistym systemem w plików z którego będą odczytywane treści plików. (**rozbudować**)

Rmdir

Operacja ta pozwala na usuwanie etykiet z wirtualnego systemu plików. (**rozbudować**)

Unlink

Umożliwia usuwanie obiektów z wirtualnego systemu plików. Wszystkie klasy obiektów mogą byc usuwane: pliki, etykiety, atrybuty(?).

```
1 rm /vhfs/@music/milan.1.mp3
```

(**rozbudować**)

Getattr

Operacja pozwala na pobranie atrybutów obiektu w kategoriach atrybutów takich jak pliki i katalogi tradycyjnego systemu plików.

Fgetattr

Ekwiwalent powyższej operacji, ale dla plików.

Opendir

Nic ciekawego tutaj się raczej nie dzieje - trzeba podać dobry handler.

Nieimplementowane operacje

(**Wymienić zbiór operacji, które są oferowane przez Fuse, ale nie zaimplementowane + powody tego**)

4.3.3 Automatyczna ekstrakcja metadanych

(**Napisać o tym jak to ręczne używanie nowego interfejsu jest w sumie jednak kłopotliwe i lepiej byłoby posiadać narzędzia ekstrakcji danycn.**) Libferris? "Serwis" dla innych aplikacji (nawet tych legacy).

4.3.4 Spójność, aktualność danych

(**Napisać o libInotify**)

4.4 Testowalność

Poziomy testowania - testy jednostkowe, testy komponentowe, testy integracyjne

4.5 Projekt implementacji

4.5.1 Architektura aplikacji

DIAGRAM KOLABORACJI

- Aplikacja
  - Analizator Składni (ang. *lexer, scanner*)
  - Parser
  - Interpreter
  - Menadżer metadanych (MetadataManager)
- Abstrakcja na libInotify (obiekt pełnomocnika (ang. *proxy*) InotifyService)
- Abstrakcja fuse (obiekt pełnomocnika FuseService)
- Abstrakcja bazy danych (DatabaseService - używa biblioteki SQLAlchemy)

4.5.2 Narzędzia

Biblioteki

- PLY
- pyInotify
- SQLAlchemy
- Elixir
- SQLite

Zręby (ang. framework)

- unittest
- Mox (biblioteka do automatycznego tworzenia mocków)
- BASH

4.5.3 Model danych

File, Attribute, Tag, View

```
1 ln -s /@music/@limit[10]/@order.by[@name].using[@strlen]
```

4.5.4 Gramatyka języka

Specjalne ciągi znaków

Są to ciągi znaków spełniające takie same warunki jak identyfikatory, tyle że rozpoczynają się od znaku tyldy zamiast `@`.

```
1 @music.children = @music.children[]
```

4.5.5 Poprawność typologiczna

Poprawność typologiczna jest sprawdzana przez obiekt klasy TypeChecker.

4.5.6 Interpretery zapytań

Ze względu na dynamiczną naturę zapytań, oraz jego deklaratywność wyrażenia ścieżkowe są interpretowane przez obiekt klasy interpretera, który tłumaczy je na język działającej pod spodem bazy danych.

Ewaluacja drzewa AST

AST (ang. *Abstract Syntax Tree*) - abstrakcyjne drzewo składni.

Nested

Signature Typy sygnatur:

- binder

```
1 /vhfs/@music
2 /vhfs/@music/@f.limit[10]
3 /vhfs/@music/@f.order[@desc].by[@name].using[@strlen]
4 /vhfs/@music/@f.order.by[@name].using[@strlen]
```

- method

```
1            /vhfs/@music.children
2            /vhfs/@music.children[]
```

- constructor - ?
- operator - binary operators, unary operators, casting operators
- value

(**opis sposobu ewaluacji**)
Ewaluacja zapytania jest wykonywana poprzez klasę interpretera. Interpreter implementuje wzorzec projektowy wizytora (ang. *visitor*)

4.5.7 Prezentacja wyników zapytań

(**translacja wynik zapytania -¿ wynik serwowany przez Fuse**)

Rozdział 5

Rozszerzenia VHFS

5.1 Skojarzenia obiektów

Tworzenie ascocjacji pomiędzy plikami podobnymi

5.2 Plugins

Narzędzia do wyłuskiwania metadanych o różnych rodzajach plików.

5.3 Alternatywne narzędzia systemowe

Część rozwiązań została przyjetą ze względu na komptybilność wsteczną z istniejącymi narzędziami systemowymi. Mogłoby okazać się, że implementacja alternatywnych narzędzi jest najlepszym sposobem na spopularyzowanie nowego podejścia do systemu plików.

- *vcp* (cp)
- *vmv* (mv)
- *vln* (ln)
- *vls* (ls)

5.4 GUI

Projekt alternatywnego dostępu do metadanych - poza tradycyjny interfejs systemu plików VHFS mógłby udostępniać API w popularnych językach programoania tak aby możliwe było korzystanie z innych dobrodziejstw bazy danych.

5.5 I18n

Wsparcie dla wielu języków - zamiast:

```
1 /@muzyka.children
```

mogłoby być:

```
1 /@muzyka.dzieci
```

```
1 /@muzyka/@jazz/@f.sortuj[@f.@rosnąco].po[@nazwa].używając[@f.porównanie]
```

dodatkowy interfejs

Rozdział 6

Podsumowanie

6.1 Opis stanu projektu

Stan jest niedokończony, do remontu.

6.2 Przyszłość projektu

Przepisanie prototypu na język ze statyczną kontrolą typów i wczesnym wiązaniem nazw (C++ wydaje się być odpowiedni).

6.3 Pytania otwarte

Technologia bazodanowa, czy też inne rozwiązanie, które będzie działać "pod spodem".

Skorowidz

Bibliografia

[bit09] History of storage systems, 6 2009.

[DN65] R. C. Daley, P. G. Neumann. A general-purpose file system for secondary storage. *AFIPS '65 (Fall, part I): Proceedings of the November 30–December 1, 1965, fall joint computer conference, part I*, strony 213–229, New York, NY, USA, 1965. ACM.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns*. Addison-Wesley, Boston, MA, January 1995.

[GJSO91] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, James O'Toole. Semantic file systems. *SOSP*, strony 16–25, 1991.

[Lov04] Robert Love. *Kernel Linux - przewodnik programisty*. Helion, 2004.

[RT74] Dennis Ritchie, Ken Thompson. The unix time-sharing system. *Commun. ACM*, 17(7):365–375, 1974.

[SG96] Abraham Silberschatz, Peter B. Galvin. *Podstawy systemów operacyjnych*. Wydawnictwo Naukowo-Techniczne, 1996.

[Ste02] W. Richard Stevens. *Programowanie w środowisku systemu UNIX*. Wydawnictwo Naukowo-Techniczne, 2002.