Porównanie Unixowych systemów plików

Maria Fronczak

15. stycznia 2004

1 Wprowadzenie

Poniższa prezentacja jest częścią prezentacji porównującej Unixowe systemy plików, przygotowanej przez Andrzeja Awramiuka, Marię Fronczak i Mateusza Zakrzewskiego. Obejmuje zagadnienia ogólne, pseudo-systemy plików i rozproszone systemy plików. Lokalne systemy plików są omówione w prezentacjach pozostałych osób z zespołu.

1.1 Definicje

By móc zajmować się porównywaniem Unixowych systemów plików, należy najpierw określić, co to jest Unixowy system plików. Jednak podanie precyzyjnej definicji jest naprawdę trudnym zadaniem.

Według [4] **system plików** to mechanizm przechowywania informacji i dostępu tak do danych, jak i do programów systemu operacyjnego.
System plików składa się zazwyczaj z dwóch wyraźnie wyodrębnionych części:

- zbioru plików
- struktury katalogów - za pomocą której organizuje się i udostępnia informacje o wszystkich plikach danego systemu

Czasem w niektórych systemach plików jest jeszcze trzecia część - strefy czy partycje, które fizycznie lub logicznie wyodrębniają większe zbiory katalogów.

Unixowy system plików natomiast, wg. [1], to zbiór plików i katalogów o następujących cechach:

- ma katalog root (/), który zawiera pozostałe katalogi i pliki

- każdy plik i katalog jest jednoznacznie identyfikowany przez swoją nazwę i katalog, w którym jest umieszczony - oraz przez jednoznaczny identyfikator, zwany inode'em
- jest niezależny od innych systemów plików

Systemy plików Unixa działają na dwóch podstawowych obiektach: plikach i katalogach, przy czym katalogi też są plikami - choć o specjalnej budowie. Dlatego też jednym z zagadnień, które jest omawiane w części prezentacji poświęconej lokalnym systemom plików, jest reprezentacja pliku.

Podana powyżej definicja Unixowych systemów plików wydaje się być stworzona głównie do opisu lokalnych systemów plików; definicja ta nie jest odpowiednia dla wielu systemów plików, które są uznawane za Unixowe, często nawet (jak specfs) są wręcz istotne dla pracy samego systemu. Nie bardzo jednak widać, jaka definicja obejmowałaby wszystkie, nawet najdziwniejsze, Unixowe systemy plików. Poprzestanę więc na podanej jako podstawie porównań, dodatkowo rozważając następujące zagadnienia:

- sposób trzymania danych na dysku (dla systemów lokalnych, czyli nie w mojej części prezentacji). Dla systemów, które będę opisywać będą to wręcz różne miejsca przechowywania danych lub nawet brak składu dla danych
- czy na systemie plików daje się wykonać tradycyjne Unixowe polecenia dla systemów plików, takie jak mount; jeśli tak, to czy zachowanie systemu plików jest wówczas „typowe" (wcale nie zawsze zawartość katalogu, na którym jest montowany system plików jest nadpisywana, czasem pozostaje w jakiś sposób dostępna).
  Może warto przy okazji wspomnieć, że nie wszystkie systemy plików muszą być montowane na katalogu. System specjalny namefs, obsługujący łącza nienazwane, można montować na pliku zwykłym.
- „Sposób współpracy" z VFS-em - np. jakie operacje na plikach bądź całym systemie plików są udostępniane, jak wyglądają prywatne dane dla pliku trzymane w v-węźle
- czy system plików zachowuje Unixową semantykę operacji dla plików, w szczególności semantykę współdzielenia pliku (czyli porządkowanie operacji w czasie, przy odczycie przekazywana jest zawsze najbardziej aktualne zawartość) - to będzie oceniane raczej dla systemów rozproszonych
- wydajność

1.2 Typy Unixowych systemów plików

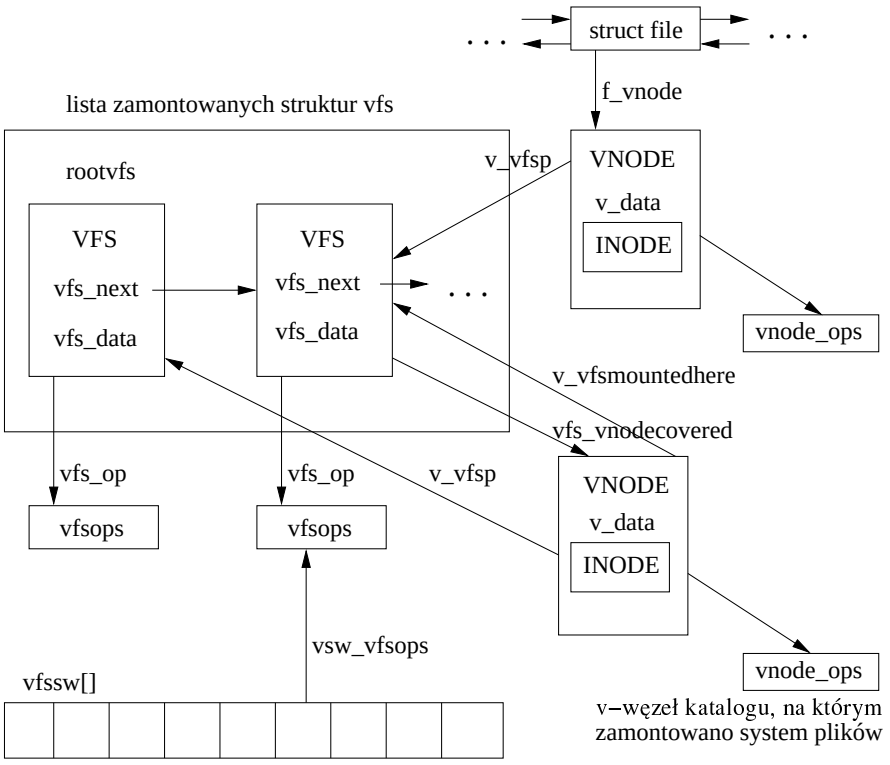
Kiedyś Unixowe systemy plików były „dyskowe", to znaczy pliki trzymane były na lokalnym dysku, bądź oparte na RAM-ie (taki system plików trwał do rebootu systemu), jednak same koncepcje implementacji były w obu wypadkach podobne.
Przez ostatnie kilkanaście lat pojawiły się jednak systemy plików innego typu - pseudo-systemy plików, które dla użytkownika wyglądają (zazwyczaj, co będzie dalej omówione) jak „zwykłe" systemy plików, są jednak implementowane na zupełnie inne sposoby.
Są wreszcie rozproszone systemy plików, będące rozproszoną implementacją klasycznego modelu systemu plików.

Systemy lokalne (ext3, XFS, ReiserFS, JFS) omówione są w prezentacjach przygotowanych przez Andrzeja Awramiuka i Mateusza Zakrzewskiego.
W tej prezentacji omawiam inne przykładowe systemy plików, które zakwalifikować można do dwóch grup: pseudo-systemy plików (w tym systemy oparte na RAM-ie) i rozproszone systemy plików.

1.3 Architektura systemów plików

Architektura systemów plików nie wydaje się może na pierwszy rzut oka bezpośrednio związana z zagadnieniem porównania różnych Unixowych systemów plików. Jednak to właśnie architektury systemów plików pozwalają na korzystanie z wielu różnych systemów plików pod jednym chodzącym Unixem (na początku przecież Unix był systemem z jednym systemem plików, który rozwinął się w obecny s5fs).
Poza tym architektura systemów plików wymusza na systemach plików zgodność z pewnymi wzorcami, np. rodzajami operacji możliwych do wykonania na systemie plików czy na plikach w systemie. To właśnie zgodność z tymi wzorcami - czyli np. które operacje i jak system plików implementuje, jakie prywatne dane są trzymane w v-nodzie - będzie jednym z kryteriów porównania systemów plików.

Architekturą, którą krótko przedstawię dla potrzeb dalszych części prezentacji, jest architektura VFS/vnode, pozwalająca na korzystanie z wielu różnych systemów plików. Daje ona ogólny interfejs do systemu plików i plików, implementując te jego części, które są od niezależne od typu systemu plików i pozostawiając implementację części specyficznych samemu systemowi plików.



Na potrzeby tej prezentacji wystarczy powiedzieć, że systemy plików reprezentowane są przez struktury vfs, zawierające pola vfs_data, wskazujące na dane specyficzne dla systemu plików i pole vfs_op, wskazujące na strukturę vfsops, czyli listę operacji na systemie plików, które dany system udostępnia (np. mount czy statfs). Zbiór takich operacji jest określony, ale system plików nie musi implementować wszystkich (co zostało przedstawione w sekcji „Podsumowanie").
V-węzeł natomiast jest strukturą związaną z konkretnym plikiem (otwartym), zawierającą pola niezależne od konkretnego systemu plików (np. v_type - typ pliku) i zależne od niego: pole v_data wskazuje na prywatne dane, zależne od systemu plików (dla lokalnych systemów plików jak ext3 będzie to zwykle inode, dla NFS rnode, dla specfs snode itd.), pole v_op wskazuje zaś na strukturę vnodeops, która udostępnia listę operacji możliwych do wykonania na pliku w tym systemie (znów jest określona długa lista możliwych operacji, zależnie od systemu nie wszystkie muszą być implementowane).

Tablica vfssw jest tablicą wkompilowanych systemów plików - wpis dla danego systemu zawiera m.in. dowiązanie do operacji na tym systemie i funkcję inicjującą dla tego systemu plików.

2 Pseudo-systemy plików

2.1 Wstęp

Co do niektórych pseudo-systemów plików można się zastanawiać, czy nazwa „system plików" nadal jest wobec nich uzasadniona. Jednak wszystkie przedstawione poniżej pseudo-systemy plików są nazywane systemami plików, są też rdzennie Unixowe. Poza tym są ciekawe i pokazują, jak różnie mogą wyglądać Unixowe systemy plików.

2.2 specfs

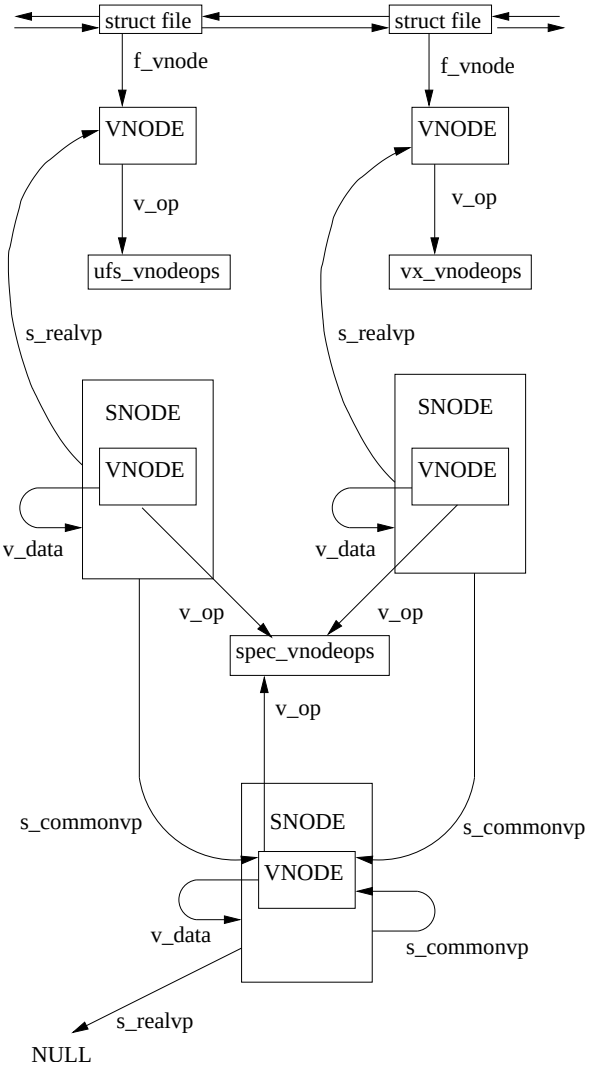
specfs - specjalny system plików - ilustruje, jak bardzo pseudo-systemy plików mogą być niezgodne ze sformułowaną definicją Unixowego systemu plików. specfs jest uznawany za system plików o tyle, że np. ma wpis w tablicy systemów plików, jednak użytkownicy są całkowicie nieświadomi jego istnienia - w szczególności nie mogą dostać informacji o nim poprzez statfs, nie mogą go montować ani nijak bezpośrednio z niego korzystać.
specfs daje interfejs do urządzeń przez pliki specjalne, interfejs, który może być stosowany przez dowolny inny system plików wspierający pliki specjalne. Pozwala na synchronizację dostępu do urządzeń. Na poziomie jądra daje metodę tłumaczenia v-node'a reprezentującego plik specjalny na v-node reprezentujący urządzenie.

Każdy v-węzeł związany z plikiem specjalnym ma odpowiadający mu towarzyszący węzeł pliku specjalnego: s-węzeł (s-node od specfs node, czasem rozwijane jest także jako „shadow node" - czyli węzeł przesłaniający), przechowujący prywatną kopię informacji związanych z urządzeniem (np. numer następnego bajtu do odczytania, v-węzeł potrzebny do sterowania tym urządzeniem). Po co taki dodatkowy węzeł? Potrzebny jest dlatego, że v-node pliku specjalnego i v-node urządzenia mają różne zbiory operacji: v-węzeł związany z plikiem specjalnym używa struktury vnodeops systemu plików, w którym istnieje ten plik specjalny; v-węzeł związany z urządzeniem używa zaś spec_vnodeops, której operacje działają na urządzeniu. Operacje, których potrzebujemy przy dostępie do urządzenia to te działające na urządzeniu. Daje nam to s-node, co więcej, operacje z systemu plików, w którym znajduje się plik specjalny nadal są dostępne przez „zwykły" v-node tego pliku.

Oczywiście wiele różnych plików specjalnych może być odwzorowanych na to samo urządzenie, dlatego na potrzeby synchronizacji dostępu urządzenie jest reprezentowane za pomocą wspólnego s-węzła (każdy s-węzeł zwią-

zany z tym urządzeniem pamięta taki wspólny s-węzeł w polu s_commonvp). Przy pierwszym otworzeniu pliku specjalnego powstaną więc dwa s-node'y: wspólny i związany z danym plikiem specjalnym, natomiast przy kolejnych już tylko s-węzły związane z plikami specjalnymi.

Na poniższym rysunku obie stuktury file związane są z plikami specjalnymi (jednym w UFS, drugim w Veritas fs), które to pliki specjalne są związane z tym samym urządzeniem.



Przekierowanie operacji z operacji specyficznych dla systemu plików na operacje na urządzeniu odbywa się następująco: gdy wywoływana jest operacja na pliku specjalnym sprawdzane jest, że jest to plik specjalny, po czym w tablicy haszującej (po numerach major i minor urządzeń) znajdowany jest

snode tego pliku specjalnego, z którego mamy już dostęp do właściwych operacji na urządzeniu (spec_vnodeops, do których mamy dostęp z snode'a, przekierowują na operacje pobrane z odpowiedniej pozycji w odpowiedniej tablicy rozdzielczej dla urządzeń). Osobna tablica haszująca dla snode'ów potrzebna jest dlatego, że w v-węźle pliku specjalnego nie ma miejsca na dowiązanie do snode'a (jak widać na rysunku), istnieje tylko dowiązanie w drugą stronę, od snode'a do vnode'a pliku specjalnego.

W podobny sposób działa fifofs, także wykorzystując metodę węzłów przesłaniających - fifonode (kolejki fifo są reprezentowane przez ich własny system plików, właśnie fifofs, korzystający z implementacji przez strumienie). Ze względu na podobieństwo do specfs nie będę jednak dokładniej omawiać fifofs.

2.3 Systemy „informacyjne"

Pod tą dość nieudolnie wymyśloną nazwą rozumiem systemy, które nie mają żadnego fizycznego składu dla informacji, które udostępniają - są to zwykle jakieś informacje systemowe. Poza dostępem do informacji takie „informacyjne" systemy plików dają też możliwości operowania na zasobach, do których zapewniają interfejs poprzez operacje na plikach.

Jeśli chodzi o zgodność tego typu systemów z podawanymi wcześniej definicjami systemów plików, w szczególności Unixowych, widać, że jest pod tym względem zupełnie nieźle - jest właściwa struktura katalogów, niezależność od innych systemów plików, zaimplementowane są operacje na plikach i systemie plików. Jedynie zawartości plików nie są przechowywane, ale wyznaczane na żądanie użytkownika, a zaimplemetowane operacje na plikach - np. zapis - nie są typowym zapisem do pliku, pozwalają za to wykonać operacje na zasobach systemu operacyjnego.

2.3.1 procfs

Systemu /proc nie będę opisywać dokładnie ze względu na to, że był już omawiany na zajęciach laboratoryjnych. System ten jest interfejsem do przestrzeni adresowej działających procesów, w szczególności umożliwia odczyt i zmianę zawartości procesu czy wykonanie operacji sterujących procesem za pomocą standardowego interfejsu systemu plików.

2.3.2 processor file system

System plików procesora udostępnie interfejs do poszczególnych procesorów w architekturze SMP. Zamontowany jest w katalogu /system/procesor, każdy procesor jest reprezentowany przez jeden plik w tym systemie. Nazwy plików są dziesiętnymi reprezentacjami numerów procesorów, są to pliki stałego rozmiaru tylko do odczytu, zawierające informacje m.in. o stanie procesora (aktywny, odłączony), jego typie, szybkości czy rozmiarze pamięci podręcznej. Pliki reprezentujące procesory stanowią „informacyjną" część systemu, natomiast możliwość operowania na poszczególnych procesorach daje plik ctl, tylko do zapisu (i tylko przez administratora) - zapis do niego pozwala np. odłączyć procesor.

2.4 Systemy oparte na RAM-ie

Jak napisałam, systemy oparte na RAM-ie powstawały już dosyć wcześnie. Łatwo się domyślić, że nie są systemami trwałymi, żyją tylko do rebootu systemu. Napisałam też, że wczesne implementacje lokalnych dyskowych systemów plików i systemów bazujących na RAM-ie nie różniły się zbytnio: zwykle po prostu lokalne systemy plików były instalowane w wydzielonym obszarze RAM-u (RAM-dysku). Oczywiście na taki system trzeba było poświęcić sporo RAM-u, m.in. dlatego powstały systemy zaprojektowane tylko pod RAM, bardziej efektywne. W sekcji tej omówię krótko dwa takie systemy: MFS i tmpfs.

Odrębności systemów opartych na RAM-ie związane są raczej ze miejscem przechowywania plików i sposobem wykonywania operacji na nich, poza tym jednak wyglądają jak „zwykłe" Unixowe systemy plików, z właściwą strukturą katalogów, operacjami na systemie plików i plikach, semantyką operacji na plikach itd.

2.4.1 Cechy systemów opartych na RAM-ie

Systemy oparte na RAM-ie są oczywiście systemami tymczasowymi, przydają się więc raczej tam, gdzie trwałość danych nie jest tym, na czym nam zależy (np. aplikacje korzystające z plików tymczasowych). Można by uznać, że nie są potrzebne osobne systemy oparte na RAM-ie, że pliki tymczasowe - w „zwykłych", lokalnych systemach plików - i tak nie zdążą być zapisane na dysk, bo szybciej będą usuwane - zawsze jednak pozostaje kwestia tworzenia i usuwania plików, co będzie wymagać dostępu do dysku - i będzie czynnikiem spowalniającym.

Tak więc motywacją do powstania systemów opartych na RAM-ie była potrzeba posiadania systemu plików, w którym operacje tworzenia plików i dostępu do nich byłyby bardzo szybkie. Pierwszym krokiem w tym kierunku było użycie RAM-dysków, na których można było utworzyć „zwykły" system plików (np. s5fs czy FFS), jednak wadą takiego rozwiązania było duże zapotrzebowanie na pamięć fizyczną i słabe wykorzystanie zasobów systemowych. Dlatego utworzono systemy plików, efektywniej korzystające z RAM-u.

2.4.2 MFS

Memory File System (University of California w Berkeley), szerzej opisany w pracy [5] jest systemem opartym na RAM-ie, w budowie zbliżonym do FFS-a, czyli szybkiego systemu plików z Berkeley (nazywanego też ufs-em). Co więcej, cały system plików tworzy się w przestrzeni adresowej procesu, który obsługiwał operację jego tworzenia: - tworzenie i montowanie MFS-a wygląda następująco:

- Wołamy newfs z odpowiednimi parametrami. W przestrzeni adresowej newfs alokowane jest miejsce pod system plików, następnie wołane jest mount dla MFS-a.

- mfs_mount() tworzy vnode reprezentujący „urządzenie blokowe", jakie będzie udawać pamięć przydzielona pod system w przestrzeni adresowej newfs; vnode taki zawiera pid procesu newfs i wskaźnik do pamięci przeznaczonej pod system plików. Inicjalizowana jest też lista żądań wejścia/wyjścia do tworzonego systemu plików.

- Wywoływane jest mount dla systemu plików ufs, z utworzonym vnode'em jako parametrem. Wykonywana jest operacja mount z ufs-a, z tym że zamiast na dysku lokalnym system jest montowany w przestrzeni procesu newfs.

Po zakończeniu montażu proces newfs nie wraca z mount (mfs_mount), zostaje w jądrze, czekając na zlecenia do zamontowanego systemu.

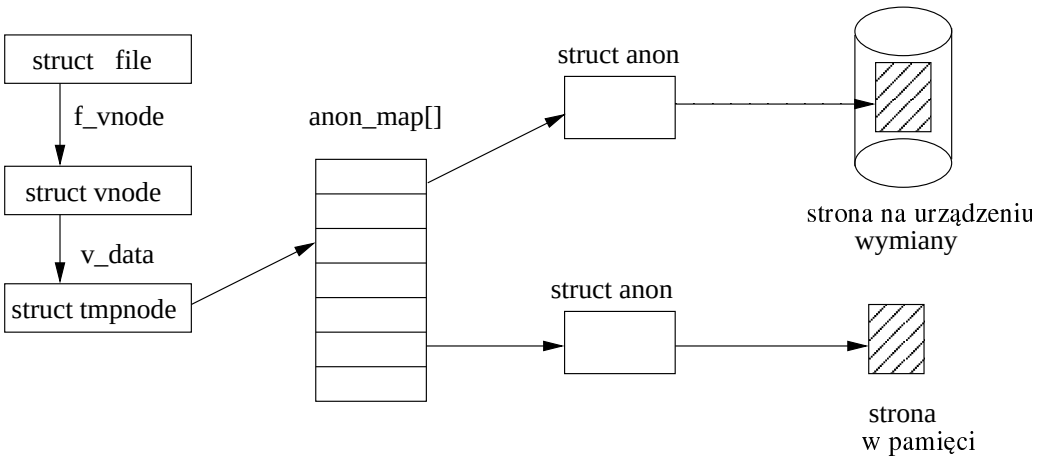
Operacje w tym systemie są obsługiwane przez ufs, chyba że jest to jakaś operacja wejścia/wyjścia. W takim wypadku zlecenie wstawiane jest do kolejki zleceń systemu MFS i proces newfs jest budzony. Po obsłudze zlecenia znów zasypia.
Odczyt czy zapis z pominięciem newfs zdarza się w dwóch sytuacjach: przy mount i unmount (przy odczycie i zapisie superbloku).

Skoro system jest w całości w pamięci wirtualnej procesu montującego, jego strony mogą być wyswapowane; strony MFS rywalizują też po prostu o pamięć z innymi procesami. Dzięki mechanizmowi wymiany stron system plików może być większy niż wynika to z ograniczenia pamięci procesu newfs.

Wadą MFS-a jest, że wymaga dwóch przełączeń kontekstu przy każdej operacji (ze względu na obecność pośredniczącego procesu newfs) jak również, że system jest w odrębnej od przestrzeni jądra przestrzeni adresowej, czyli zapis czy odczyt wymaga kopiowania z jednego miejsca w pamięci w drugie - między pamięcią jądra a pamięcią systemu (poza nieuniknionym kopiowaniem z czy do przestrzeni użytkownika). Podwójne kopiowanie jest wadą tym bardziej, że kopiowania zajmują najwięcej czasu (np. przy zapisie kopiowanie od użytkownika do jądra i z jądra do systemu plików, czyli do przestrzeni newfs, zajmuje ok. 90% czasu CPU). Mimo to wzrost wydajności operacji na plikach w MFS - w porównaniu do operacji na plikach w ufs na dysku lokalnym - jest znaczny: MFS jest około dwa razy szybszy przy operacjach czytania/pisania i wielokrotnie szybszy przy operacjach dotyczących metadanych (np. przy tworzeniu pliku).

2.4.3 tmpfs

Z wymienionymi wadami MFS-a radzi sobie tmpfs Suna, korzystający z mechanizmów pamięci wirtualnej systemu V. tmpfs jest w całości w jądrze (nie ma żadnego procesu pośredniczącego jak przy MFS), przy czym wszystkie plikowe metadane są w pamięci niestronicowanej (dla dodatkowego zwiększenia wydajności), przydzielanej dynamicznie, zaś bloki z danymi w pamięci stronicowanej, reprezentowane są przez udogodnienie stron anonimowych podsystemu pamięci wirtualnej SVR4.



Każda strona anonimowa odwzorowywana jest za pomocą obiektu anonimowego (anon) w położenie strony w pamięci fizycznej lub na urządzeniu wymiany. Każdy węzeł tmpnode ma wskaźnik do mapy - tablicy odwzorowującej strony pliku na obiekty anon.

Oczywiście strony w pamięci (czyli bloki danych) mogą być wyswapowane, system musi też rywalizować o pamięć. Brak dodatkowego kopiowania czy procesu pośredniczącego (co występowało w MFS) sprawia, że tmpfs jest jeszcze szybszy niż MFS - porównianie tmpfs i lokalnego ufs, opisane w pracy [6] mówi o około dziewięciokrotnie większej szybkości w tmpfs przy operacjach typu zapis i nawet kilkadziesiąt razy większej dla np. tworzenia i zamykania plików (przy tych operacjach na pustym pliku jest to ok. 35 razy).

2.5 Systemy-„nakładki”

Systemy, które opiszę w tym punkcie, nazwałam nakładkami, ponieważ są swego rodzaju nakładkami na inne systemy plików. Charakteryzują się tym, że zamontowane na jakimś systemie plików nie przesłaniają systemu, na którym są zamontowane i umożliwiają dostęp do plików w tymże systemie (choć czasem mogą w jakiś sposób ten dostęp modyfikować, tak że operacje wykonane na podstawowym systemie plików za pośrednictwem systemu-„nakładki” wyglądają mają inny efekt niż wykonane bezpośrednio w podstawowym systemie plików; mogą też być jakieś efekty uboczne operacji wykonanych przez „nakładkę”).

Systemy „nakładkowe” mają zwykle „normalną” Unixową strukturę katalogów, są zgodne z główną podaną definicją cech Unixowego systemu plików. Można zastanawiać się tylko, czy są w pełni niezależne od innych systemów plików - odpowiedzią jest chyba, że nie, ale przynajmniej nie są uzależnione od konkretnych innych systemów plików, są nakładkami, które można nałożyć na wiele różnych innych systemów plików.

2.5.1 Systemy bezstanowe

Niektóre systemy-„nakładki” można określić jako bezstanowe - nie mają „własnych” plików, można przez nie wykonać tylko operacje na plikach w systemie, na którym są zamontowane. Ale nawet o plikach tego systemu bezstanowy system-nakładka nie trzyma żadnych informacji, nie implementuje też zwykle własnych operacji na plikach. Opiszę krótko dwa przykłady systemów bezstanowych: nullfs i crossfs.

Nullfs pozwala zamontować poddrzewa hierarchii plików (w szczególności z różnych systemów plików) w jednym miejscu, dając drugą nazwę ścieżkową dla każdego pliku z takiego poddrzewa. Pozwala to np. pozbierać wszystkie katalogi jednego użytkownika, porozrzucane po różnych systemach plików, w jednym katalogu w nullfs. Przy wykonywaniu operacji na plikach pod nullfs w większości używane są operacje z pierwotnego systemu plików, z którego dany plik pochodzi.

Crossfs zamontowany na jakimś systemie plików również daje dostęp do plików systemu, na którym jest nadmontowany. Nie udostępnia operacji na plikach, wszelkie operacje przekierowuje na operacje systemu, na którym jest zamontowany. Jedyne, co robi, to wykonuje jakąś akcję, gdy przy rozwiązywaniu nazwy ścieżki przekraczany jest punkt jego zamontowania - zazwyczaj jest to wypisanie do logów systemowych komunikatu z uid i gid procesu, który wykonuje operację w obrębie crossfs. Pozwala to sprawdzać, kto wykonywał operacje w katalogu, na którym zamontowano crossfs.

2.5.2 Systemy wielowastwowe

Systemy wielowarstwowe są systemami montowanymi zazwyczaj nad systemem plików tylko do odczytu i nadbudowują nad nim warstwę (lub warstwy) pozwalające na zapis - w szczególności możliwy jest dostęp do podstawowego systemu plików i modyfikacja istniejących w nim plików.

TFS Translucent FS Suna jest systemem plików dostarczającym mechanizmów kontroli wersji, dość szczególnym systemem lokalnym. Pozwala on na nadbudowanie wielu warstw nad podstawowym systemem plików, na którym jest zamontowany. Każdy katalog składa się w nim z wielu warstw, łączących się ze sobą przez pliki ukryte - dowiązania wyszukiwania (searchlinks). Każda warstwa jest kolejną wersją katalogu, zewnętrzna jest najnowszą (możliwe są też warstwy wariantowe). Sięgnięcie do starszych warstw odbywa się przez przejście po dowiązaniach.
Zapis możliwy jest tylko w warstwie zewnętrznej, stosowane jest więc kopiowanie przy zapisie (copy-on-write): jeśli chcemy zmodyfikować plik z którejś z niższych warstw, jest on kopiowany do warstwy zewnętrznej i dopiero wtedy modyfikowany. Podobnie można usunąć plik z niższych warstw - wstawiana jest wtedy w warstwie zewnętrznej zaślepka oznaczająca, że plik jest usunięty i by go nie szukać w niższych warstwach (oczywiście żadne prawdziwe usuwanie w warstwach niższych nie jest wykonywane).

union fs (union mount) zapewnia nieco podobną funkcjonalność: daje łączny obraz systemów plików zamontowanych pod warstwą union mount, podobnie możliwe jest modyfikowanie plików tylko w wierzchniej warstwie, stosowany jest mechanizm copy-on-write i zaślepek przy usuwaniu (możliwe jest też dostęp do pliku usuniętego poprzez pomijanie takich zaślepek).

3 Systemy rozproszone

Zagadnienia związane z systemami rozproszonymi omawiane były już na wykładzie, podobnie jak rozproszony system plików NFS. Dlatego też pozwolę sobie pominąć definiowanie pojęć związanych z systemami tego typu, jak również opis NFS-a. Systemami, które omówię, będą RFS, AFS i powstały na postawie AFS-a DFS.

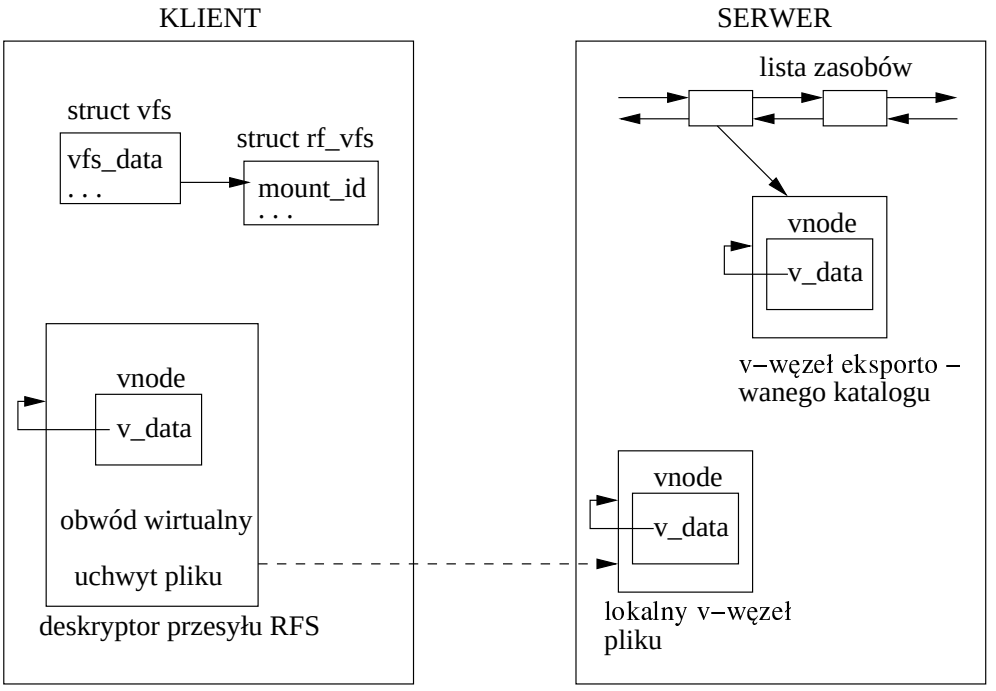
3.1 RFS

RFS, czyli Remore File Sharing (AT&T), jest systemem znacznie mniejszej popularności niż NFS, używany był w SVR3, wyszedł z użycia, choć SVR (i tylko SVR) nadal go wspiera. Znalazł jednak miejsce w tej prezentacji jako najbardziej chyba Unixowy z rozproszonych systemów plików - zachowuje bowiem pełną Unixową semantykę operacji na plikach (co zwykle przy systemach rozproszonych się nie udaje), jak również pozwala np. na dostęp do plików specjalnych. Zaimplementowany jest z wykorzystaniem podsystemu VFS. Jest niezależny od konkretnej techniki sieciowej, inaczej niż NFS.

Niestety RFS umożliwia udostępnianie plików tylko między maszynami z systemem Unix, jest to jednak cena, za którą uzyskano zgodność z standardami Unixowego systemu plików. Zachowana jest Unixowa semantyka współdzielenia plików (czyli po zapisie do pliku zmiany są widoczne również dla innych klientów już przy następnych odczytach). RFS zapewnia również sieciowy dostęp do plików specjalnych i łączy nazwanych oraz pozwala na zakładanie blokad na plikach. Pozwala również na odwzorowanie identyfikatorów UID i GID.

Każdy eksportowany katalog ma przydzieloną symboliczną nazwę zasobu (scentralizowany serwer nazw mapuje nazwy zasobów na ich położenia sieciowe); dla każdego takiego katalogu istnieje pozycja w liście zasobów jądra z nazwą zasobu, wskaźnikiem do vnode'a korzenia, wskaźnikiem do listy klientów uprawnionych do dostępu i wskaźnikiem do listy montażu u klientów

Jest stanowy, trzyma tablicę wszystkich klientów. W wypadku awarii klienta serwer anuluje informację o jego stanie, być może zmniejsza liczbę odwołań do i-węzłów odpowiednich plików, zwalnia blokady itp.



(obwód wirtualny jest obiektem opisującym sposób połączenia sieciowego i jest ustanawiany przy mount. Deskryptor przesyłu to prywatne dane RFS-a, trzymane w v-nodzie)

Zwiększenie szybkości dostępu do plików uzyskiwane jest przez stosowanie pamięci podręcznej po stronie klienta. Oczywiście ukrywane jest przed klientem prawdziwe położenie pliku. Pamięć podręczna u klienta działa następująco: w wypadku zapisu najpierw następuje zapis do pamięci podręcznej, dopiero potem na serwer. W wypadku, gdy więcej klientów ma otworzony plik, do którego nastąpił zapis (takie informacje o klientach są trzymane), serwer wysyła im komunikat unieważnienia ich pamięci podręcznej; ich pamięć podręczna będzie znów włączona po ściągnięciu danych z serwera Odczyt z pamięci podręcznej następuje tylko wtedy, gdy wszystkie dane do odczytu są w (nieunieważnionej) pamięci podręcznej.
Cache'owanie części plików jest jednak jedyną zastosowaną formą poprawy wydajności, np. już przy rozwiązywaniu nazwy ścieżki pliku nie jest stosowana żadna optymalizacja, nazwa jest rozwiązywana po stronie serwera, przy

czym jeśli wyjdzie poza punkt zamontowania RFS (przez ".."), rozwiązywanie nazwy wraca do klienta.

3.2 AFS

AFS, Andrew File System, powstał na Carnegie Mellon University jako system przeznaczony do obsługi dużej liczby użytkowników. Jest systemem łatwo skalowalnym. Zastosowane w nim mechanizmy mają zmniejszać ruch w sieci i odciążać serwer, zwiększając wydajność i szybkość operacji.

Jeśli chodzi o ogólną budowę systemu, składa się on z komórek (cells) - grup serwerów zarządzanych razem. Każdy serwer zarządza swoim zbiorem woluminów (typowo woluminem może być np. katalog jednego użytkownika). Woluminy mogą być przemieszczane między serwerami, by wyrównać obciążenie sieci. Baza danych położeń woluminów trzymana jest na każdym serwerze. Maszyny klienckie są odróżniane od serwerów, maszyna nie może być jednocześnie serwerem i klientem.

By zmniejszyć ruch w sieci i zwiększyć wydajność stosowane jest cache'owanie po stronie klienta, przy czym w pamięci podręcznej klienta trzymane są nie tylko dane (części plików), ale także metadane: informacje katalogowe, dowiązania symboliczne. Inaczej niż np. w RFS klient sam dokonuje rozwiązania nazwy ścieżki, nie odwołując się do serwera. Kopiowanie do klienta odbywa się w porcjach po 64 KB - biorąc pod uwagę, że zazwyczaj zachodzi lokalność odwołań, pozwala to umieścić z wyprzedzeniem dane w cache'u klienta.

AFS odpowiada podanej definicji Unixowego systemu plików, natomiast nie zapewnia Unixowej semantyki współdzielenia plików - stosowana jest semantyka sesji, to znaczy uaktualnianie danych na serwerze następuje nie przy operacji write, ale przy close. Klient pobierając dane z serwera dostaje „callback promise" - obietnicę powiadomienia, gdyby dane w jego pamięci podręcznej stały się nieaktualne. Klient posiadający taką obietnicę może modyfikować plik w swoim cache'u, nie wysyłając zmian na serwer. Zmiany są wysyłane, gdy klient zamyka plik, wtedy też serwer wysyła powiadomienie do innych klientów, mających obietnicę dla zmodyfikowanego pliku (mówi się wówczas, że obietnica tych klientów jest złamana). Jak widać, serwer jest stanowy (czyli trzyma dane o klientach).

3.3 DFS

DFS powstał jako rozwinięcie AFS. Nie ma w nim już rozróżnienia na serwery i komputery klienckie, jedna maszyna może być i serwerem, i klientem. W odróżnieniu od AFS, daje lepszą semantykę współdzielenia (zachowana jest ścisła semantyka Unixowa) i gwarancję spójności.
Do ciekawszych szczegółów implementacji należą gwarancje ważności danych wyrażane za pomocą żetonów: zarządca żetonów przyznaje i odbiera klientom żetony. Żetony należą do jednego z czterech rodzajów:

- danych (tu są żetony odczytu i zapisu na przedział bajtów w pliku, zabranie żetonu do odczytu oznacza, że trzeba odczytać nowe dane z serwera, posiadanie żetonu do zapisu pozwala zapisać w pamięci podręcznej, bez przesyłania do serwera)
- stanu
- blokad (póki klient ma takie żetony, nie musi się z nikim porozumiewać co do blokad na pliku)
- otwarcia (typy otwarcia)

Jak widać, zastosowanie żetonów pozwala uzyskać spójność danych i Unixową semantykę operacji.

4 Podsumowanie

Unixowe systemy plików zostały przedstawione i porównane ze względu na kilka warunków:

- posiadanie struktury katalogowej z nadrzędnym katalogiem /:
  - systemy posiadające taką strukturę: np. dyskowe (ext3, ReiserFS)
  - systemy „płaskie” jak bfs - startowy system plików, zawierający programy niezbędne do uruchomienia Unixa (SVR4) - będący prostym, płaskim systemem plików z jednym katalogiem, zawierającym tylko zwykłe pliki
  - zupełny brak struktury katalogowej - jak specfs
- niezależność od innych systemów plików:
  - niezależne (ext3, MFS, procfs)

  - nakładki na inne systemy, modyfikujące dostęp do systemu plików, na którym są zamontowane: Translucent fs, union fs
  - nakładki bezstanowe, przekazujące operacje systemowi, na którym są zamontowane: nullfs, crossfs
  - nakładki na poziomie vnode'ów, bez samodzielnej struktury systemu plików: specfs, fifofs

- różne sposoby trzymania danych na dysku (systemy lokalne): np. jedna kopia superbloku (ufs) bądź wiele (s5fs)

- różne miejsca trzymania danych:
  - dysk: ext3, ReiserFS, JFS
  - RAM: MFS, tmpfs
  - pobieranie danych na bieżąco i przedstawianie ich w postaci plików: procfs, processor fs

- wydajność: wzrost wydajności w systemach tymczasowych (względem systemów lokalnych), techniki stosowane w systemach tymczasowych (system w przestrzeni procesu lub w przestrzeni jądra). Stosowanie cache'owania (tylko dane lub także metadane) jako sposób podnoszenia wydajności w systemach rozproszonych.

- różne dane trzymane w prywatnych danych (vnode'a):
  - tradycyjne inode'y: ufs, ext3,...
  - węzły przysłaniające: specfs, fifofs

- montowalność:
  - montowalne z przysłonięciem katalogu-punktu zamontowania - ext3,
  - montowalne, bez przysłonięcia - union mount fs
  - montowalne na pliku - namefs
  - niemontowalne - specfs

- różne implementowane operacje na samym systemie plików, np.:
  - mount - implementują: ufs, bfs, nfs, procfs; nie implementują: specfs, fifofs
  - mountroot (zamontowanie jako główny system plików) - implementują: ext3, ufs, nie implementują: procfs, bfs, nfs, specfs

  - ...

- różne implementowane operacje na plikach:
  - mkdir/rmdir - implementują: ufs, ext3, nie implementują: bfs
  - create/remove - implementują: ext3, bfs, MFS, nie implementują: procfs
  - symlink - implementują: ext3, nfs, nie implementują: procfs, bfs
  - ...

- zgodność z Unixową semantyką operacji na plikach:
  - zgodne: ext3, MFS, RFS
  - niezgodne: AFS

Literatura

[1] S. Pate: UNIX Filesystems: Evolution, Design, and Implementation. Wiley Publishing, 2003.

[2] Vahalia U.: Jądro systemu Unix. Nowe horyzonty. Wydawnictwa Naukowo-Techniczne Warszawa, 2001.

[3] Goodheart B., Cox J.: Sekrety magicznego ogrodu. Unix system V Wersja 4 od środka. Podręcznik Wydawnictwa Naukowo-Techniczne Warszawa, 2001.

[4] Silberschtz A., Galvin P.: Podstawy systemów operacyjnych. Wydawnictwa Naukowo-Techniczne Warszawa, 2002.

[5] McKusick, M., Karels, M., Bostic, K.: A Pageable Memory Based Filesystems. Proceedings of the Summer 1990 USENIX Technical Conference, 1990

[6] Snyder P.: tmpfs: A Virtual Memory File System. Proceedings of the Autumn 1990 European UNIX Users' Group Conference, pages 241-248, 1990