CAM 4 - Fast File System

Olaf Barthel

Coś starego, coś nowego: Fast File System, przeszłość, teraźniejszość, przyszłość - opowieść w stylu gotyckiego horroru - autor: Olaf Barthel

Kiedy używasz swojej Amigi niekoniecznie zdajesz sobie sprawę, że jest coś takiego jak system pilków, który znajduje zastosowanie we wszystkich operacjach wykonywanych przez system i oprogramowanie. Jest praktycznie niewidoczny (oczywiście do czasu, gdy coś pójdzie nie tak jak powinno). W tym artykule postaram się, aby system plików stał sie bardziej "widoczny". Opiszę jego dziwną i zakręconą historię, jego ograniczenia i obecny stan w kontekście systemu AmigaOS4.0.

Struktury danych i szczegóły dotyczące implementacji elementów nie będą poruszane w dokładny sposób gdyż mogłoby to zanudzić czytelnika. Przygotujmy się na wstępne, techniczne poruszenie tematu.

1. Tło historyczne

1.1 Pochodzenie

To co znamy jako system plików Amigi, to część DOSa (disk operating system), która została przeportowana do oryginalnego AmigaOS w roku 1985, gdy zrezygnowano z planów implementacji CAOSa - Commodore Amiga Operating System. Ten system operacji dyskowych zwany był Tripos i była to komercyjna wersja eksperymentalnego systemu operacyjnego stworzonego w Laboratoriach Komputerowych Uniwersytetu w Cambridge (w Anglii). Tripos posiada pewne cechy Unixa: powstał w zamyśle portowalnego wielozadaniowego systemu operacyjnego, w który wbudowano portowalny język programowania (BCPL). Faktem jest, że ten portowalny język programowania stał się prekursorem języka C, języka, który został zaimplementowany w systemie Unix (przez uproszczenie języka BCPL, stworzono język B, którego rozwój sprawił narodzenie się języka C; nie, wcale tego nie zmyślam).

W internecie nie ma zbyt wielu informacji na temat tego co sprawiło, że Tripos stał się tym, czym się stał. Artykuł opublikowany w 1979 roku , autorstwa Dr Mike'a Richardsona, który pracował na tym systemie, jest wszystkim do czego mogłem sie odwołać.

Kilka szczegółów na temat AmigaDOS. Niektórzy ludzie wciąż wierzą, że rdzeń systemu operacyjnego Amigi wykorzystuje kod Triposa. Nie jest to prawdą. Wszystko co zostało "zapożyczone" z Triposa to system plików i jego działanie oraz funkcjonalność, taka jak API, Shell i jego standardowe komendy (które są w zasadzie programami). Portowalny kernel Triposa został dostosowany dla Amigi przez Dr Tima Kinga, który zdobył prawa, aby przekształcić system operacyjny w produkt komercyjny. Z tym pomysłem zgłosił się do Metacomco Ltd., obecnie nieistniejąca firma informatyczna mieszcząca się w Bristol, w Anglii.

1.2 Implementacja

Oryginalny system plików Amigi, dostarczany wraz z pierwszymi Amigami w roku 1986, pracował tylko na dyskietkowych nośnikach. Rok, dwa lata później potrafił również obsłużyć większe nośniki, takie jak twarde dyski. O tym jednak później. Do tego jednak czasu, użytkownicy Amigi musieli przyzwyczaić się do dziwnych dźwięków wydawanych przez stacje dysków, które brzmiały jakby dyskietki były przecinane na pół. Ten dźwięk, nazwany później "gronking", w zasadzie powodowany był przez zaprojektowane i zaimplementowane struktury danych systemu plików.

"Gronking" emitowany był przez poruszanie się głowicy stacji dysków. Powodem tego, było to, że system plików bardzo fragmentował dane. To wskazuje na wiele niedogodnień tego projektu: nie został zaprojektowany z myślą o prędkości. Z drugiej jednak strony, można stwierdzić, że był zaprojektowany z myślą o poprawności danych. Wszystkie dane i

CAM 4 - Fast File System

Olaf Barthel

struktura danych systemu plików były chronione przy pomocy sum kontrolnych i zwalniania struktur danych, co czyniło możliwym odzyskiwanie danych z uszkodzonych dysków. "Zdecentralizowany" układ struktury danych na dysku również sprawiał, że było mało prawdopodobne, aby jeden błąd doprowadził do zniszczenia wszystkich danych na dysku (oczywiście nie przesądza to, że taka sytuacja nie mogła nastąpić).

Kolejna sprawa, z którą wiązała się "zdecentralizowana struktura danych system plików" to "fragmentacja", która niszczyła efektywność systemu plików. Co więcej, każdy używany przez system plików blok, był chroniony przez sumę kontrolną. To oznaczało, że każdy musiał być odczytywany osobno, jego suma kontrolna sprawdzana a jego zawartość "wyładowywana". Efektem tego było relatywne bezpieczeństwo danych, które jednak było nierelatywne do prędkości. W rzeczywistości słabe punkty w kodzie odpowiedzialnym za cache bloku sprawiały, że z im większą ilością bloków danych system musiał sobie poradzić w cache, tym wolniej całość pracowała.

1.3 Integracja z systemem operacyjnym

Inaczej niż w tradycyjnych systemach operacyjnych, system plików w systemie operacyjnym Amigi nie jest częścią samego kernela systemu operacyjnego. System plików to proces przy pomocy którego oprogramowanie i uzytkownik komunikują się wymieniając między sobą wiadomości. Wiadomości wysyłane są pośrednio przez API (Application-programming Interface) znane jako dos.library, które oddziałuje pomiędzy kernelem systemu plików Triposa, systemami plików i amigowym oprogramowaniem.

Projekt przekazywania sobie wiadomości posiada swoje wady i zalety. Jedną z wielu zalet jest to, że z racji iż format i układ wiadomości może na przestrzeni lat ulec zmianom, API systemu plików może być na bieżąco rozszerzane. Ta cecha nie istniała od samego początku . Dodano ją później, zapewniając systemowi ogromną elastyczność. Projekt stwarzał również możliwości asynchronicznej implementacji urządzeń I/O, która przebiegała prosto i efektywnie. To z kolei sprawiało, że wielozadaniowość systemu stawała się coraz silniejszą jego stroną.
Główna wada znajduje się w strefie rozmiarów i jednakowego podejścia. W wielozadaniowym systemie operacyjnym rozkład zadań w kernelu gwarantuje duże tolerancję w stosunku do jednakowego traktowania każdej informacji. System plików musi jednak przeanalizować nadchodzące wiadomości i upewnić się, że każdy klient traktowany jest w identyczny sposób. To spore wyzwanie zwłaszcza, że jedne zadania systemu plików mogą trwać dłużej lub krócej, podczas gdy inne zadania mogą przejmować całkowitą kontrolę nad systemem plików.

Taki projekt systemu plików jest bardzo wyjątkowy i unikalny. Dla przykładu, pod Unixem, aplikacja w celu wykonania operacji na plikach, stosuje w kierunku kernela zawołanie, które skutkuje wywołaniem kodu systemu plików. Takie podejście sprawia, że operacje systemu plików poddają się normalnym praktykom stosowanym przez inne zadania. Unixowy system plików, najbliższy amigowego, stosuje się w systemie Sun Microsystems i nazywany jest Networked File System (NFS). Serwer NFS musi rozwiązywać takie same problemy jak amigowy system plików: żądanie klienta przychodzi jako wiadomość, która musi być traktowana tak samo jako inne. Typowym rozwiązaniem tego problemu jest podzielenie pracy na "nici", każda zmierzająca do żądania innego klienta. Tak w zasadzie działał oryginalny system plików: wykorzystywał szczególną cechę Triposa zwaną "threading", która działała na zasadzie przełączania kontroli pomiędzy różnymi procedurami, każdą wykonywaną w swojej kolejności, ze swoim własnym stosem i lokalnymi zmiennymi (to zdaje się być dziełem Algol 68, z którego BCPL oddziedziczył tę cechę; dla przykładu, podobną funkcjonalność można znaleźć w Modula-2). Trudno było zrobić uczciwy rozkład zadań przy istniejącym zamyśle, ale zadziała - do pewnego stopnia.

1.4 Przyspieszając system plików: FFS

W okolicy 1986-87 roku, wady istniejącego systemu plików stały się zbyt oczywiste. Prędkość pozostawiała wiele do

życzenia a pierwsze twarde dyski, które były dostępne dla Amigi nie mogły przekraczać rozmiaru 50MB dla partycji; system plików po prostu nie potrafił bezpiecznie poradzić sobie z partycjami większych rozmiarów z powodu jego zaprojektowania. Nałożono ograniczenie na rozmiar struktury danych, która śledziła zapotrzebowanie na miejsce przez katalogi i pliki.

Te i inne ograniczenia prowadziły do rozwoju tego co znamy jako Fast File System, rozszerzona wersja oryginalnego systemu plików. Steve Beats, jeden z inżynierów Commodore, został wysłany do Metacomco, aby przyjrzeć się działaniu obecnego systemu plików i wymyśleć usprawnienia. To co naprawdę zrobił, to zmodyfikował struktury danych i przepisał cały system plików do czystego asemblera MC68000. Andy Finkel, wówczas głównodowodzący komórki Commodore pracującej nad rozwojem oprogramowania systemu operacyjnego powiedział mi, że naprawdę spodziewali się, że Steve Beats przygotuje wersję napisaną w języku C, która byłaby bardziej portowalna. Od tamtej pory, Amiga została pobłogosławiona lub przeklęta nieportowalnym, trudnym w modyfikacji domyślnym systemem plików, który jakby na to nie patrzeć działał duzo szybciej niż oryginał.

Zmiany dokonane w strukturze danych były raczej niewielkie, miały jednak ogromny wpływ na efektywność. Porzucono element sum kontrolnych w blokach danych co umożliwiło systemowi plików skupić się na dużo większej liczbie bloków, które sterownik potrafił odczytać/zapisać podczas jednego kroku, bez potrzeby jakiejkolwiek pomocy czy nadzoru. Stało się to możliwe dzięki wykorzystaniu DMA (Direct Memory Access) w kontrolerach twardych dysków, które umożliwiają aplikacjom wczytywanie danych do pamięci bez większego obciążenia procesora. System plików próbuje spakować dane w następujące po sobie bloki, zmniejszając fragmentację. Aby przeskoczyć ograniczenie 50MB dla partycji, nowa struktura danych została dodana, co umożliwiło uzyskanie większej ilości miejsca. I wreszcie ostatnie, ale nie najgorsze, struktura danych w katalogach została uporządkowana, sprawiając, że głowica stacji dysków poruszała się w poprzek nośnika w stale obranym jednym kierunku. Uniknięto w ten sposób skakania głowicy tam i z powrotem (co sprawiało, że stacja dysków "chrząkała").
Wszystkie te zmiany nie zamieniły domyślnego systemu plików w Amidze w wysoce efektywny projekt, ale znacznie zmniejszyły uciążliwości.

1.5 Zwiększenie użyteczności, operacje network oraz internacjonalizacja

W kolejnych dwóch latach, Commodore pracowało nad ulepszoną wersją systemu operacyjnego Amigi. W jego nowej wersji pojawiła się wspomniana, przepisana do asemblera implementacja system plików, który miał obsługiwać więcej operacji niż oryginał. Wprowadził również pomysły takie jak blokowanie rekordów (record locking) oraz zmiana notyfikacji. Te usprawnienia stały się możliwe, dzięki projektowi elastycznej implementacji przekazywanych wiadomości do systemu plików.

Znowu dokonano zmiany struktury danych i wprowadzono grupę oraz właściciela ID dla wszystkich plików i katalogów na dysku. Ta cecha nie była wykorzystana w systemie plików, lecz w warstwie sieciowego systemu plików (NFS), który był częścią pakietu Commodore's Envoy.

Dalsze usprawnienia polegały na zmianie algorytmów i struktur danych zaangażowanych w odnajdywanie nazw plik i katalogów. Szukając nazwanego obiektu, nazwy zawsze były porównywane pod względem wielkości znaków. Jednakże, oryginalny system plików wiedział tylko jak porównywać znaki w czystym ASCII. Gubił się, gdy przychodziło do porównania znaków w Amiga ISO 8859 Latin 1. Dla przykładu, oznacza to, że dla oryginalnego systemu plików słowo "FACADE" było takie samo jako "facade", ale już słowa "FAÇADE" i "façade" nie były.

Ostatnią rzeczą była dodana obsługa twardych i miękkich "połączeń". To było trochę ryzykowne przedsięwzięcie, z racji, że oryginalny projekt i powiązane z nim API nie było wyposażone w taką funkcjonalność. Miękkie połączenie

znane jest jako symboliczne połączenie na Unixowych systemach operacyjnych. Wskazuje na nazwę pliku, która zawiera dane odnoszące się do połączenia. Połączenie twarde bezpośrednio wskazuje dane odnoszące się do niego.

1.6 Poprawa efektywności

Podczas gdy na przestrzeni kolejnych kilku lat pojawiły się kolejne ulepszenia API, jedyna większa zmiana w systemie plików nastąpiła wraz z wprowadzeniem DCFS - Directory Caching File System. Zaprojektowany został przez kolejnego z inżynierów Commodore - Randalla Jessupa.

DCFS próbował zwiększyć prędkość skanowania katalogów. Aby odczytać zawartość katalogu, system plików musi "odwiedzić" wiele połączonych bloków, każdy z nich identyfikuje pojedyncze miejsce wejścia do katalogu. Bloki mogą być rozrzucone po całym systemie plików. Projekt FFS próbuje adresować ten problem poprzez alokacację wejść do katalogu leżących w okolicy root bloku, który był umieszczony w okolicy środka partycji, zmniejszając przez to ruchy głowicy. Nadal jednak mogło nastapić zjawisko fragmentacji, które kierowało by głowicę z początku na koniec nośnika próbując odczytać punkt wejścia kolejnego katalogu. DCFS ulepszał również i to przez przechowywanie zawartości katalogu na drugiej liście, która zawierała indywidualne nazwy i metadata wejść katalogów. Przez przechowywanie tych danych blisko siebie, katalogi mogły być odczytywane a ich zawartość raportowana poprzez przeszukiwanie 2-3 bloków zamiast 20-30. Druga lista, zwana "directory list", mogła w rzeczywistości zawierać ograniczone informacje. W przypadku systemu plików, takie ograniczenie oznacza zazwyczaj kłopoty.

Ograniczenie wprowadzone przez DCFS nie było niewielką cenę jaką trzeba było zapłacić za zalety, które przyniosło. Wprowadziło to do systemu plików modyfikacje narażające go na większą złożoność oraz wrażliwość na błędy. Poprzednio jedna próba zapisu była wystarczająca do uaktualnienia metadata i gwarantowała integralność systemu plików. Teraz, trzy dodatkowe próby były wymagane. Jeśli jeden z tych kroków został pominięty z powodu zawieszenia systemu lub resetu, cały system plików można było szybko przywrócić do poprzedniego stanu. Z DCFS nie chodziło tylko o pobranie informacji, które bloki były wykorzystywane, ale należało również odtworzyć "directory list", a to mogło podwoić lub potroić czas przywracania systemu do poprzedniego stanu. Narażało to również system plików na częstsze występowanie błędów.

Aby poprawić efektywność, Rnadall Jessup dodał opcję zwiększenia domyślnego rozmiaru bloku wykorzystywanego przez system plików. Do tej pory mógł być tylko jedne sektor na blok, a sektor mógł mieć rozmiar 512 bajtów. Po zmianach, można było łączyć wiele sektorów (2, 4, 8, 16) w jeden blok.

Fakt napisania systemu plików w asemblerze, sprawił, że ustabilizowanie DCFS z ulepszeniami trwało dosłownie lata. Oryginalny pomysłodawca nigdy go nie ukończył gdyż Commodore zbankrutowało. Dalszym rozwojem pokierował prawie sześć lat później Heinz Wrobel.

1.7 Obecne zmiany i obecny stan prac

Większość zmian w obecnym kodzie systemu plików dokonał Heinz Wrobel. Wszystko było głównie nakierowane na przekroczenie bariery 4GB jako rozmiar partycji. System plików nic nie wie na temat rozmiaru i położenia partycji w kwestii tego ile bajtów jest w to zaangażonwanych. Zna tylko numery bloków i opiera się na najniższych warstwach w translacji pomiędzy tymi numerami bloków oraz na tym, czego oczekuje urządzenie pamięci masowej. W tym przypadku API urządzenia pamięci masowej oczekują offsetów bajtów. Te z kolei ograniczane są co do rozmiaru i położenia partycji do maksymalnie 2^32 bajtów (około 4.2 biliona bajtów) lub dokładniej, 4GB. Heinz Wrobel zaadaptował warstwę umożliwiającą dostęp do bloku systemu plików, która wykorzystuje szerokie, 64-bitowe komendy dostępy, które umożliwiają nośnikowi wykorzystać obszary większe niż 4GB. Dodatkowo wprowadzono tuzin poprawek,

których poprzedni "właściciel kodu" nie był w stanie zastosować.

1.8 Ograniczenia

Wybór struktury danych definiuje ograniczenia implementacji. Najpierw proste rzeczy: nazwa wolumenu, pliku czy katalogu nie może być dłuższa niż 30 znaków. Dla miękkich "połączeń", nazwa obiektu do którego następuje połączenie jest ograniczona przez liczbę bajtów w bloku; dla 512-bajtowego bloku oznacza to, że docelowa nazwa połączenia nie może być dłuższa niz 288 znaków (praktyczne ograniczne, w rzeczywistości jest mniejsze: 255 znaków). Ilość plików i katalogów, które mogą być przechowywane na dysku ograniczona jest wyłącznie poprzez ilość wolnego miejsca; obecnie nie istnieje ograniczenie co do liczby wejść, które mogą prowadzić do katalogu. Maksymalny rozmiar nośnika jest uzależniony od rozmiaru sektora; 232 sektory mogą być dostępne, a każdy sektor może liczyć co najmniej 512 bajtów. Maksymalny rozmiar pliku jaki może zostać bezpiecznie uzyty to 2^31 bajtów, czyli około 2GB.; to ograniczenie wynika z faktu, że wszystkie wielkości używane przez system plików są zdeklarowane jako liczby 32-bitowe a plik, którego rozmiar będzie większy niż 2GB mógłby wyskoczyc poza ramy i mieć rozmiar ujemny. Re-implementacja FFS specjalnie narzuca ograniczenie rozmiaru pliku do 2GB i nie pozwoli na stworzenie większego pliku.

2. Struktury danych w skrócie

Dla osób niezainteresowanych w technicznych aspektach systemu plików proponuję przeskoczyć ten rozdział.

Struktury danych używane przez system plików Amigi są bardzo dobrze udokumentowane, z kilkoma wyjątkami. Pierwszy z nich to ten, że nie są w ogóle udokumentowane w oryginalnej dokumentacji Commodore, ale nadrobione to jest z nawiązką w dokumentacji "3rd party" opublikowanej przez programistę Ralpha Babela. Drugi wyjątek to struktury danych DCFS, które z tego co wiem, nigdy nie zostały udokumentowane publicznie. Zamknięta grupa programistów miała do nich dostęp, włączając w to twórców oprogramowania "ratunkowego". Moje informacje na temat DCFS oparte są na danych, które udało się zebrać innemu programiście, Holgerowi Kruse.

2.1 Dziedzictwo BCPL

Mało już dzisiaj znany język programowania BCPL miał dosyć osobliwe podejście do tematu adresowania danych. BCPL wyciągał z podstaw najmniejsze, adresowalne jednostki pamięci poprzez definiowanie ich jako słowo. Dla procesora MC68000, rozmiar słowa wynosił 4 bajty lub 32 bity. Prowadziło to do kilku dużych struktur danych używanych przez system plików, takich jak data przechowująca takie informacje jak aktualna czas i dzień z 96 bitową precyzją. Powiązane struktury danych były bardziej prymitywne niż w C. W języku BCPL powiązane struktury danych posiadają podobną nazwę dla tablicy z wartościami indeksów nazwanej tablicy. Skoro już o tym mowa, metoda dostępu do struktury danych umożliwiała prosty polimorfizm w kontekście tego, że żadne sprawdzanie typu danych nie było ani nie mogło być wykonane. Jest to jednak dozwolone.

2.2 Katalogi i metadata

Każdy punkt wejścia do katalogu i każdy katalog wykorzystują wirtualnie tę samą strukturę danych: w zapytaniu istnieje nazwa obiektu, jego data modyfikacji, atrybuty (do odczytu, do zapisu, wykonywalny itd.), jego rozmiar (dla pliku lub twarde połączenie do pliku) i lista liczby bloków. Ta lista wskazuje miejsce wejcia do pierwszych kilku bloków, które zawierają plik lub tworzą rozsypaną tablicę katalogów. Wejścia do katalogów przechowywane są w podłączonych listach, których pierwszy punkt wejścia zarezerwowany jest w tablicy katalogów. Do którego slotu katalog wejścia zostanie przekierowany zależy od wartości nazwy wejścia. Liczba wejść wchodzących do bloku zależy od jego

rozmiaru. Dla bloku o rozmiarze 512 bajtów, wykorzystywane są 72 wejścia. Maksymalna długość nazwy pliku ustawiona jest na 30 znaków.

W przeciwieństwie do unixowego systemu plików, katalogi i wejścia do katalogów nie są dwoma osobnymi elementami. Podczas gdy w unixowym systemie plików plik katalogu jest kojarzony z nazwą pliku zawierającą liczbę inode. Inode zawiera metadata i odnosi się do danych. Amigowy system plików zachowuje nazwę i metadata razem w blokach wejścia katalogu. Sprawiło to, że implementacja "połączeń" w amigowym systemie plików stała się dużo bardziej skomplikowana niż w unixie, gdzie "połączenie" jest kolejnym katalogiem wejścia odpowiadającym inode.

2.3 Przechowywanie danych

W przeciwieństwie do unixowego systemu plików, gdzie hierarchia struktury danych odpowiada blokom danych, z których składa się plik, w amigowym systemie plików oparte jest to na odnośnikach do połączonych list bloków danych. Amigowy system plików nie zna koncepcji rozmiaru. Tablica odwołuje się do każdego pojedynczego bloku. W oryginalnym projekcie, bloki danych zawierały coś więcej niż dane. Zawierały także sumy kontrolne i informacje, które łączyły się z pojedynczymi blokami.

2.4 Połączenia

Twarde połączenia są prawie tym samym co wejścia do katalogów, za wyjątkiem, że nie posiadają żadnych informacji w tablicy bloków, która jest częścią wszystkich bloków wejść do katalogów. Z trudem odnoszą się do liczby bloków obiektu do którego są podłączone. Dla każdego twardego połączenia, połączenie wraca z podłączonego obiektu do pierwszego twardo podłączonego bloku. Kilka twardych połączeń odpowiadających temu samemu obiektowi również przechowywane są na liście połączeń. Miękkie połączenia sa podobne do twardych, w których nie wykorzystywana jest tablica bloków. Zawierają również nazwę obiektu do którego są podłączone.

2.5 Listy katalogów

Wprowadzone w DCFS listy katalogów zawierają wirtulanie tą samą informację, która jest przechowywana w blokach wejść katalogów. Zbierane są w listy, które pakują metadata i nazwy plików. Informacja przechowywana jest w podłączonych listach bloków katalogów.

2.6 Zarządzanie przechowywaniem danych

W amigowym systemie plików informacje na temat tego na których blokach są umieszczone dane i metadata systemu plików przechowywane są w podłączonej liście bloków, w której każdy bit w każdym bajcie odpowiada blokowi na partycji. Zanim nastąpi jakakolwiek operacja zapisu, musi nastąpić poprawne ustawienie mapy bitowej, aby przez przypadek żadne z danych nie zostały nadpisane. Proces odtwarzania mapy bitowej zwany jest walidacją. Za każdym razem, gdy mapa bitowa jest niezwalidatowana, a może to nastąpić przez niedokończony zapis spowodowany powieszeniem systemu lub świadomym jego wyłączeniem, system plików musi powtórnie zostać zwalidatowany. Jest to dosyc długotrwały proces.

2.7 Podsumowanie

Jeżeli jeszcze nie zauważyłeś, amigowy system plików jest starym projektem, który w ogromnym stopniu dba o bezpieczeństwo danych, bardziej niż o efektywność. Inne systemy wywodzące się z innych źródeł (np. the Berkeley

CAM 4 - Fast File System

Olaf Barthel

Fast File System wprowadzony w 4.2BSD Unix w 1983 roku) wprowadzały wiele ulepszeń, zwłaszcza w kwestii prędkości. Amigowy system plików w tym czasie nie przeszedł tylu ulepszeń. Istnieje wiele wad projektu, które niekoniecznie są dobrym pomysłym aby był on rozwijany w kolejnych wersjach systemu operacyjnego. Z drugiej jednak strony całkowita zmiana jest prawie niemożliwa.

3. Reimplementacja Amiga Fast File System i inne dziwne pomysły

3.1 Dlaczego na pierwszym miejscu postawiliśmy reimplementację?

Z zaprezentowanych informacji trudno wyciągnąć wnioski inne niż to, ze byłoby złym pomysłem kontynuowanie prac nad amigowym systemem plików. Powinno się go zastąpić czyms lepszym. Na nieszczęście ludzie, którzy doszli do tych wniosków, zawiedli w przygotowaniu odpowiedniego zamiennika. Typowe założenia powodowały stworzenie prostego, ciągłego systemu, który pracował szybko w przypadku prostych aplikacji, lecz nie do końca bezpiecznie. Systemy plików również gubią dane. Dzieje się tak z powodu założenia, że prawdopodobnie bezpieczniejsza konstrukcja systemu plików nie może stracić czy uszkodzić danych - podobny sposób rozumowania jak to, że skoro Titanic był niezatapialny nie potrzebował łodzi ratunkowych.

Wolny i ograniczony jest amigowy system plików mimo, że posiada bardzo dobre narzędzia do odzyskiwania danych jak i dla celów defragmentacji. Należy to do rzadkości w przypadku systemów plików pisanych przez tzw. 3rd party. Sposób w jaki działa amigowy system plików jest bardzo dobrze zrozumiały i obecnie użytkownicy są bardzo dobrze zaznajomieni z jego zachowaniem i ubytkami. Są to niebywale dobre powody do tego, aby nadal kontynuować pracę nad jego rozwojem. Nie zaspokajają jednak potrzeby lepszego systemu plików. Implementacja lepszego systemu plików wymaga czasu, powiązanego z dużym nakładem pracy i testów. Wynika z tego jasno, że reimplementacja jest zadaniem dużo prostszym. I jest tego prosty powód: sprzęt Amigi nowej generacji musi posiadać możliwość uruchamiania programów (w trybie emulacji) pracujących na oryginalnym systemie plików napisanym w asemblerze. Istnieje również duża szansa, że kod napisany dla celów reimplementacji systemu plików będzie mógł zostać ponownie wykorzystany dla przyszłych, ulepszonych systemów. No i ostatnia rzecz, system plików, który potrafi odczytywać dane zapisanych na starych nośnikach, w starszych systemach, zawsze jest mile widziany w systemie oparacyjnym Amigi.

Ostatnie pytanie jakie się nasuwa, to "dlaczego nie przeportować już jakiegoś istniejącego systemu plików jak np. ReiserFX ze świata Linuxa?" Odpowiedź leży w strukturze systemu, która jest różna w Linuxie/Unixie i a AmigaOS. Unixowe systemy plików są typowo jednowątkowe ponieważ rozkład zadań kernela zajmuje się operacjami wykonywanymi przez system plików. Amigowy system plików, musi być wielowątkowy, aby pracować właściwie i efektywnie. To są dwa różne podejścia, które nie są proste do rozwiązania. Zwykłe przeportowanie istniejącego systemu plików nie zaprojektowane dla AmigaOS stworzą produkt będacy podstandardem.

3.2 Oto kolejny śmietnik, w którym się znaleźliśmy.

Nie zapominajmy, że podczas reimplementacji amigowego systemu plików należy rozróżnić kwestię przepisania systemu do bardziej portowalnego języka, jak np. C, i kwestię uruchomienia i sprawdzenia w działaniu takiego systemu. Wiem już czego nie zrobiłem wtedy, bo założyłem, że zrobi to ktoś inny. Oto krótka lista dobrych powodów, dlaczego nie powinno się próbować reimplementować amigowego systemu plików w domu. Amigowy system plików był trudny do implementacji. Elementy odziedziczone po BCPL pozostawiały po sobie zawsze ślad. Jak już wspomniałem na początku tego artykułu, normalnie nie zwraca się uwagi na operacje wykonywane przez system plików. Dzieje się tak dlatego, gdyż zakłada się, że działa on prawidłowo i wykonuje to co do niego należy, czyli przechowuje informacje do których chcemy miec dostęp i chroni je przed uszkodzeniem. Oczywiście doprowadzenie systemu plików do takiego etapu wymaga nakładów pracy i czasu. Większość obecnie używanych systemów plików posiada długą historię powstawania,

rozwoju i zapewniania jakości. Originalny Amiga FFS przez 12 lat przeszedł wiele. Efekt, jak widać, jest bardzo dobry. Jeśli próbujemy stworzyć nasz własny system plików, który w zamierzeniu ma zastąpić oryginalną implementację, musimy nauczyć się zasad, które obowiązują w dos.library i elementach systemu operacyjnego. Nie wszystkie z tych zasad są oczywiste i nie wszystkich można się nauczyć poprzez zwykłe przeglądanie kodu źródłowego. Również, z racji, że system plików ma być zamiennikiem obecnie istniejącego, musi być w 100% kompatybilny z oryginalnym FFS. I nie oznacza to tylko zgodności na podłożu fundamentalnym, takim jak np. w jaki sposób przechowywane są na dysku dane. Dla przykładu, przewidziano, że amigowy system plików ustawia obecny czas systemu (!) jeżeli takowy nie został ustawiony. Typowym jest, że kilka dysków wykorzystuje ten sam kod systemu plików. Aby to było możliwe, system plików musi być całkowicie otwarty. Podczas gdy dyski korzystają z tego samego kodu, żaden z nich nie może interferować z operacjami wykonywanymi przez inny. Operacje systemu operacyjnego muszą być wykonywane równolegle (wielowątkowo), lecz ani system operacyjny Amigi, ani język C, w których pisana jest reimplementacja, nie oferują tak zwanych wątków. Jeżeli próbujesz zaimplementować system plików w portowalny sposób, nie możesz polegać na specyficznych cechach sprzętu. Dla przykładu, oryginalny Workbench Amigi wykorzystywał zestaw podprocedur napisanych w asemblerze, aby zaimplementować procedurę umożliwiającą watkowość systemu plików. Nie istnieje w pełni kompletna instrukcja, w jaki sposób systen plików Amigi powinien zostać zaimplementowany. Istnieją wydzielone przykłady, niektóre z nich napisane w kilku różnych językach programowania, a w dodatku istnieje kilka różnych wersji amigowego systemu plików. Nie wszystkie są dobrze udokumentowane, a niektóre nawet nieczytelne. Nie możesz implementować tylko cech amigowego systemu plików. Musisz również zaimplementować błędy i skutki uboczne. Dla przykładu, aplikacje które oddziałują na system plików, mogą przekazywać parametry, które nie zawsze są prawidłowe. Niemniej oryginalny system plików je toleruje.

Nie brzmi to zachęcająco, prawda? I tak jest, ale to dopiero wierzchołek góry lodowej: to są cechy, które ledwie przykrywają podstawową funkcjonalność, którą tak naprawdę zajmuje się system plików. Nie poruszają ważnych elementów takich jak obsługa duzych nośników.

Praca nad reimplementacją systemu plików rozpoczęła się na początku 2001 roku. Celem było stworzenie w 100% kompatybilnego, napisanego w C, zamiennika. Zajęło to prawie dwa lata. Kod przeszedł przez prawie 200 różnych wersji, a ponad 20 testerów pomagało nam odnaleźć błędy. Z radością mogę powiedzieć, że nie wystąpiły żadne poważniejsze przypadki utraty danych. Bardzo możliwe, że było to spowodowane tym, że testerzy byli na tyle bezpieczni, że nie ufali implementacji, do momentu gdy nie stała się naprawdę dojrzała. Zostało wbudowanych kilka cech, które pozwoliły usprawnić działania systemu plików.

3.3 Tak więc, co nowego?

Czy można nauczyć starego psa nowych sztuczek? Heinz Wrobel zrobił to, gdy pracował nad systemem plików w latach 1997-1999. Były to jednak zwykłe rozszerzenia. Ale co z naszym "nowym psem", reimplementacją amigowego systemu plików?

Istnieją ograniczenia definiowane przez system plików. Struktury danych na dysku dyktują, co można, a czego nie można zrobić. Istnieją również wyjątki, w których możesz dokonywać zmian, które są raczej pożyteczne. Nowa implementacja amigowego systemu plików znacząco różni się od oryginalnej implementacji zwłaszcza w poniższych miejscach. Kolejność w jakiej zapisywane są bloki próbuje zminimalizować ryzyko uszkodzenia danych.

Za każdym razem gdy modyfikowana jest struktura systemu plików (podczas tworzenia katalogu czy zapisu do pliku), zmiany wpływają na zawartość kilku bloków na dysku. Jeżeli musimy zmienić zawartość więcej niż jednego bloku, ważnym staje się , który blok będzie pierwszy. Dla przykładu, gdy tworzysz nowy katalog, musisz ustawić blok miejsca przechowywania katalogu i musisz połączyć ten blok z katalogiem nadrzędnym. Co się stanie gdy system operacyjny

się zawiesi lub nastąpi reset zanim nastąpiła zmiana zmodyfikowanych bloków na dysk? Wygląda na to, że będzie problem, gdyż struktura systemu plików nie będzie zgodna. Co gorsze, ponieważ walidator systemu plików będzie próbował naprawić strukturę, niezgodność zostanie wykryta, a wolumen pozostanie niezapisywalny. Gdzie jest ta dyskietka z programami ratunkowymi, gdy najbardziej jej potrzebujesz?

Nowy amigowy system plików próbuje sam chronić się przed takimi wypadkami poprzez ostrożne wybieranie bloków na dysku, które w pierwszej kolejności będą zapisywane, a zapis których nastąpi w dalszej kolejności. Oryginalny system plików za bardzo nie dbał o takie szczegóły, przez co zwięszało się niebezpieczeństwo uszkodzenia.

Ulepszona notyfikacja pliku/katalogu dostarcza więcej informacji.

Aplikacja może prosić, aby system plików informował ją o zmianach jakie nastąpiły w pliku lub ogólnie, w zawartości katalogów. Poprzednio, zmiany w katalogach pozbawione były informacji co naprawdę uległo zmianie. Aby się tego dowiedzieć, aplikacja musiała ponownie odczytać zawartość katalogu i domyślić się różnic. Był to proces czasochłonny, z racji, że skanowanie katalogu nie jest jedną z najmocniejszych cech amigowego systemu plików. Ulepszona notyfikacja pliku/katalogu umożliwia nazwom i wejściom do katalogów, które zostały zmienione bezpośrednio ustalenie co uległo modyfikacji. Aplikacja może wykorzystać te informacje do optymalizacji odzewu na te zmiany. Zmiany zawartości katalogu są propagowane przez hierarchię całego systemu plików. Załóżmy, że aplikacja dokonuje zmian w katalogu, głęboko w hierarchii systemu plików a my chcemy dowiedzieć się, gdzie nastąpiła zmiana. Jak tego dokonać? Dzisiaj jedyną możliwością jest sprawdzić każdy plik i katalog w wolumenie, który został ostatnio zmodyfikowany. Dzieje się tak, gdyż system plików przechowuje tylko informacje o ostatniej dacie (dzień i czas) modyfikacji katalogu w którym nastąpiły zmiany. Ta informacja nie jest propagowana przez hierarchię systemu plików. W nowym systemie plików odbywa się to inaczej. Czasy modyfikacji pliku lub katalogu uaktualniane są wraz z całą ścieżką dostępu, aż do katalogu głównego. Flaga archiwizacji, która identyfikuje pliki lub katalogi będące w potrzebie archiwizacji również jest ukaktualniana.

System cache'owania danych jest bardziej efektywny.

System plików wykorzystuje konfigurowalną liczbę buforów wykorzystywanych do cache'owania danych znalezionych na blokach dysku, które zawierają informacje systemu plików (liczba buforów jest kontrolowana przez komendę AddBuffers). Do tej pory, liczba buforów, które przypisaliśmy nie miała większego znaczenia na ogólną poprawę pracy. Istnieją informacje, że oryginalny system plików będący częścią Kickstartu 1.0-1.3 stawał się wolniejszy, gdy musiał sprostać większej liczbie buforów. FFS radził sobie z tym dużo lepiej, lecz nadal mógł być jeszcze ulepszony.

Nowy amigowy system plików stosuje odmienne podejście dla zapewniania cache bloku z prawdziwego zdarzenia. Im więcej buforów zostanie zaalokowanych, bardziej wydajny się to stanie. Można zaalokować 500 buforów na dysk.

Nazwy plików i katalogów mogą być dłuższe niż 31 znaków.

W roku 1985, gdy nazwy plików w MS-DOS były nawet krótsze niż te stosowane w stacji dysków Commodore 1541, amigowy system plików obsługiwał do 31 znaków na pojedynczą nazwę pliku/katalogu. Było to dużo, lecz obecnie zapotrzebowanie jest na nieco więcej.

Nowy amigowy system plików obsługuje nową strukturę dysku, która umożliwia nadanie plikowi lub katalogowi nazwy składającej się z maksymalnie 107 znaków. Więcej byłoby lepiej, ale 107 znaków to maksimum na jakie umożliwia API dos.library.

CAM 4 - Fast File System

Olaf Barthel

Rozszerzenia funkcjonalności systemu plików są możliwe.

Ten pomysł nadal jest jeszcze we wczesnym stadium. Zapewnia on pewien stopień elastyczności operacyji systemu plików, którego do tej pory w Amidze nie było. Polega on na wykorzystaniu podstawowych ram systemu plików i rozszerzeniu ich przy pomocy zewnętrzenego oprogramowania, zwanego pluginami.

Funkcjonalność już zaimplementowana jako pluginy działa na poziomie bloku dysku. Zastosowanie pluginów ma miejsce przy szyfrowaniu danych i przy cache'owaniu danych. Plugin cache'u jest szczególnie użyteczny ponieważ rozszerza własny mechanizm cache systemu plików w sposób, który poprzednio wymagał pomocy programów, które musiały mądrze "załatać" sterownik. Plugin szyfrowania może również zastąpić zewnętrzne programy, które poprzednio musiały radzić sobie z podobnymi problemami jak w przypadku cache.

3.3 Kolejny krok

Po rozważeniu wszystkich rzeczy, amigowy system plików który mamy obecnie nie jest tym, co chcielibyśmy przenieść do następnej wersji systemu operacyjnego. Każda podróż wymaga dokonania pierwszego kroku: nowy amigowy system plików jest pierwszą kompletną implementacją napisaną w portowalnym języku wysokiego poziomu. To jest platforma na której będziemy bazować, zwłaszcza gdy system plików nowej generacji jest tworzony. Będzie znacząco różny od tego co znamy dzisiaj. Jeżeli są jakieś szczególne cechy, które chcielibyście, aby były zaimplementowane, napiszcie do mnie.

Tłumaczenie na podstawie Club Amiga Monthly - Sebastian Rosa.