Minix File System - niealokacja pustych bloków

Mateusz Warzyński
Uniwersytet Warszawski, Wydział Matematyki, Informatyki i Mechaniki
Warszawa, Polska
m.warzynski@student.uw.edu.pl

STRESZCZENIE

Dokument zawiera opis implementacji ograniczenia zużycia miejsca przez maszyny wirtualne oraz przyspieszenia ich działania w systemie plików Minix File System wraz z omówieniem wyników testów wydajnościowych.

1. PROBLEM

1.1 Kontekst

Pracuję w w międzynarodowej korporacji zajmującej się rozwijaniem systemu plików MFS (Minix File System) oraz wirtualizacją. Ostatnio, naszym problemem jest narastające zużycie miejsca na dysku poprzez obrazy maszyn wirtualnych. Przetrzymywanie wielu PB danych z wiedzą, że duża ich część jest niewykorzystana przez maszyny (jako puste miejsce na którym maszyna mogłaby coś zapisać), skłoniło nas do burzy mózgów na temat optymalizacji systemu plików, którego aktualnie używamy - MFS.

1.2 Rozwiązanie

Obrazy maszyn wirtualnych w dużej części są puste. MFS, gdy dostanie do zapisania blok składający się z samych zer, zapisze cały blok na dysk.

Głównym pomysłem na optymalizację przechowywania maszyn wirtualnych jest sprawienie, że nie będziemy zapisać na dysk bloków składający się z samych zer. Co więcej wykonywanie kopii dużego pliku również powinno być szybkie (kopiowane powinny być tylko niezerowe bloki).

1.3 Korzyści

- zmniejszenie ilości miejsca zajmowanego przez maszyny wirtualne
- przyspieszenie zapisu plików zawierających zerowe bloki
- przyspieszenie kopiowania plików zawierających zerowe bloki

SYSTOR '17 Warszawa, Polska

DOI: 10.475/123_4

2. ZMIANY

2.1 Implementacja

Pożądanym celem jest pomijanie zapisywania bloków, które są wypełnione tylko i wyłącznie zerami. Pozwoli to na zmniejszenie zużycia miejsca jeśli nasze pliki będą zawierały duże, ciągłe i puste części.

Aby osiągnąć zamierzony efekt musimy zmienić odpowiednią część implementacji systemu plików Minix File System. Z analizy zależności funkcji wynika, że wszystkie wywołania akcji dotyczących użycia dysku przepływają przez funkcję `rw_chunk` w pliku `/usr/src/minix/fs/mfs/read.c`. Ponadto w komentarzu w kodzie źródłowym jest napisane, że funkcja zajmuje się odczytem lub zapisem bloku. Zatem wszystkie wprowadzone zmiany powinny dotyczyć tej funkcji.

W funkcji `rw_chunk` tworzenie nowego bloku jest realizowane gdy typ bloku nie jest specjalny, blok nie istnieje oraz naszą akcją nie jest czytanie. Aby osiągnąć zamierzony cel musimy przed utworzeniem bloku w przypadku **trybu zapisu** stwierdzić, czy dane które chcemy zapisać są puste. W celu uzyskania dostępu do danych musimy je skopiować do utworzonego wcześniej **bufora** za pomocą `sys_safecopyfrom`. Następnie sprawdzić, czy którakolwiek wartość w buforze różni się od zera - co jest kosztem stałym wyznaczonym przez wielkość bloku. Jeśli nie, to możemy zatrzymać wykonywanie funkcji.

Odczytywanie danych z zerowego bloku, którego nie zapisaliśmy na dysk, działa ponieważ MFS w przypadku próby odczytu nieutworzonego bloku zwraca puste wartości. Czyli takie które byśmy normalnie mieli zapisane na dysku.

2.2 Rezultat

- **Zapis bloku** jest realizowany tylko gdy blok zawiera wartość różną od zera. Zatem po wprowadzeniu zmian pliki z relatywnie dużą liczbą zerowych bloków będą zapisywane szybciej.
- Zmiany są kompatybilne - wszystkie operacje powinny działać tak jak przed wdrożeniem.

3. TESTY

3.1 Opis

Testy wydajnościowe mają na celu sprawdzenie zmian szybkości wykonywania operacji na plikach - w szczególności zapisu i kopiowania - przed dodaniem niealokacji pustych bloków i po.

Wielkości użytych plików w testach:

- **Large** - 4000kb
- **Medium** - 400kb
- **Small** - 4kb

Typy plików:

- **Mixed** - xxxxx00000xxxxx
- **Random** - xxxxxxxxxxxxxxx
- **Zero** - 000000000000000

Gdzie x oznacza losowe dane, a 0 pustą wartość.

3.2 Zapisywanie - wyniki

Test (10 iteracji)	Przed (s)	Po (s)	Wzrost
mixed_small	0.36	0.36	0%
mixed_medium	1.91	1.31	31.41%
mixed_large	11.13	9.78	12.13%
random_small	0.2	0.21	-5%
random_medium	2.03	2.3	-13.3%
random_large	12.11	14.45	-19.32%
zero_small	0.23	0.2	13.04%
zero_medium	1.06	0.69	34.91%
zero_large	6.08	3.4	44.08%

3.3 Kopiowanie - wyniki

Test (10 iteracji)	Przed (s)	Po (s)	Wzrost
mixed_small	0.08	0.1	-25%
mixed_medium	0.15	0.16	-6.67%
mixed_large	0.31	0.35	-12.9%
random_small	0.06	0.08	-33.33%
random_medium	0.11	0.13	-18.18%
random_large	0.51	0.63	-23.53%
zero_small	0.08	0.07	12.5%
zero_medium	0.11	0.9	18.18%
zero_large	0.22	0.15	31.82%

3.4 Interpretacja

- Wzrost przy operacji **zapisu** jest widoczny dla plików posiadających bloki zerowe (mixed ma połowę bloków zerowych).

  Natomiast dla operacji **kopiowania** wzrost jest zauważalny tylko dla plików wypełnionych zerami.

- Spadek przy operacji **zapisu** jest widoczny tylko dla plików wypełnionych danymi.

  Natomiast dla operacji **kopiowania** spadek jest zauważalny dla bloków wypełnionych niezerowymi danymi lub mających połowę bloków pustych.

4. WNIOSKI

Wprowadzenie zmiany niealokowania plików przyspieszyło operacje wykonywane na plikach posiadających bloki puste przez Minix File System. Im więcej bloków pustych zawiera plik tym zmiana powoduje większy wzrost szybkości działania. Dla dużych plików może się to okazać szczególnie przydatne.

Jakkolwiek, większość plików w codziennym użyciu nie zawiera pustych bloków. A wprowadzenie niealokacji opóźnia operacje na plikach niezawierających dużej liczby pustych bloków. Zatem wprowadzona zmiana ma zastosowanie tylko gdy mamy doczynienia z potrzebą przetrzymywania dużej ilości plików z pustymi blokami.

Dla potrzeb międzynarodowej korporacji, która zajmuje się przetrzymywaniem obrazów maszyn wirtualnych, które zawierają wiele pustych bloków wprowadzenie łatki może przynieść wiele zysku z powodu ograniczenia zużycia miejsca i przyspieszenia działania tworzenia kopii zapasowych obrazów.