Systemy Operacyjne 2

Wirtualny System Plików

Arkadiusz Chrobot

Katedra Systemów Informatycznych

9 maja 2024

Plan

Wstęp

Alokator plastrowy nie jest jedynym rozwiązaniem wynalezionym przez pracowników firmy Sun Microsystem, które zostało zapożyczone w jądrze Linuksa. Innym jest *wirtualny system plików* (ang. *Virtual File System* — VFS). Jest to warstwa abstrakcji, która pośredniczy między rzeczywistym systemem plików, a resztą jądra. Dzięki niej Linux może obsługiwać wiele różnych systemów plików. Interesującym jest również to, że VFS, mimo że w całości zaimplementowany w języku C, oparty jest na modelu obiektowym. Dostarcza on ujednolicone API dla wszystkich systemów plików obsługiwanych przez Linuksa. Oznacza to, że oprogramowanie użytkowe operuje na plikach przy użyciu tych samych wywołań systemowych, takich jak `open()`, `read()`, `write()`, `close()`, niezależnie od tego jaki system plików jest zainstalowany na urządzeniu przechowującym te pliki. VFS „tłumaczy" te wywołania na operacje specyficzne dla tego systemu plików. Podsumowując: wirtualny system plików tworzy wspólny model (abstrakcję) systemu plików, który reprezentuje właściwości i operacje rzeczywistych systemów plików.

Uniksowy model systemu plików

Ten model VFS jest oparty na czterech głównych elementach oryginalnego systemu plików Uniksa: *pliku*, *katalogu*, *i-węźle* (ang. *inode* lub *i-node*") i *superbloku*. Ogólnie, system plików jest strukturą danych przechowującą informacje uporządkowane hierarchicznie. W systemach kompatybilnych z Uniksem systemy plików są *montowane* (ang. *mounted*) w określonych miejscach, nazywanych *punktami montowania* (ang. *mounting points*) wspólnego drzewa katalogowego. Tworzą one zatem wspólną przestrzeń nazw (ang. *namespace*) dostępną dla procesów użytkownika[1]. Kiedy proces użytkownika korzysta z systemu plików, to nie wie na którym urządzeniu fizycznym ten system się znajduje, w odróżnieniu do systemów MS Windows, gdzie musi wprost określić to urządzenie.

[1] W przypadku najnowszych systemów kompatybilnych z Uniksem każdy proces może mieć swoją własną przestrzeń nazw. W Linuksie ta możliwość jest dostępna od wersji 2.4.0 jądra.

Uniksowy model systemu plików

Pliki są uporządkowanym ciągiem bajtów i jednym z dwóch najważniejszych pojęć w każdym systemie kompatybilnym z Uniksem (drugim jest proces). Każdy plik ma unikatową nazwę. Procesy użytkownika mogą wykonywać na plikach takie operacje jak: otwarcie, odczyt, zapis, zamknięcie. Katalogi są plikami przechowującymi informacje (tzw. *metadane*) o innych plikach. Niektóre katalogi, nazywane *podkatalogami* są zagnieżdżone w innych. Ciągi nazw podkatalogów mogą być częściami *ścieżek*. Każdy element ścieżki (nazwa katalogu lub pliku) jest nazywany *pozycją katalogową* (ang. *directory entry* lub *dentry*). Unix obsługuje katalogi w ten sam sposób jak pliki, tzn. przy użyciu tych samych operacji. Niektóre z metadanych pliku, takich jak czas i data ostatniej modyfikacji, rozmiar, są także przechowywane w osobnych blokach na nośniku, nazywanych i-węzłami. Metadane i informacje sterujące dotyczące całego systemu plików są przechowywane w głównym bloku fizycznego nośnika, nazywanym *superblokiem*.

Uniksowy model systemu plików

Niektóre rzeczywiste systemy plików nie są zgodne z tym modelem, ale dzięki VFS nadal mogą być używane w Linuksie. Ta warstwa abstrakcji przedstawia wszystkie ich elementy w sposób umożliwiający dopasowanie do tego modelu.

Elementy Wirtualnego Systemu Plików

Kod VFS jest zgodny z zasadami programowania zorientowanego obiektowego, choć w całości napisany w języku C. Obiekty VFS to zmienne, których typy zdefiniowano z użyciem struktur reprezentujących klasy [2]. Każda z takich struktur ma pole będące wskaźnikiem do struktury wskaźników na funkcje. Wskazują one na funkcje implementujące operacje wykonywane na rzeczywistym systemie plików. Innymi słowy te funkcje to metody. VFS definiuje cztery typy obiektów: obiekty superbloku, które reprezentują superbloki zamontowanych systemów plików, obiekty i-węzłów, które reprezentują pliki w systemie plików, obiekty wpisów katalogowych (obiekty dentry), które reprezentują pojedyncze pozycje katalogowe i obiekty plików, które reprezentują otwarte pliki. Każdy obiekt danego typu ma swój własny obiekt (strukturę) operacji. Obiekt `super_operations` grupuje metody dla systemu plików, `inode_operations` grupuje metody dla plików.

[2] W języku C++ też można użyć słowa kluczowego `struct` zamiast `class` by zdefiniować klasę obiektu.

Elementy Wirtualnego Systemu Plików

Obiekt `dentry_operations` grupuje metody dla wpisów katalogowych, a obiekt `file_operations` gromadzi operacje dla otwartych plików. Niektóre z tych operacji są „dziedziczone” z grupy generycznych funkcji implementujących wspólne operacje dla wszystkich systemów plików obsługiwanych przez Linuksa.

Obiekt superbloku

Wszystkie dane o zamontowanym systemie plików są przechowywane w obiekcie superbloku. Zazwyczaj odpowiadają one informacjom umieszczonym w superbloku systemu plików w urządzeniu pamięci masowej. Istnieją jednak systemy plików utrzymywane całkowicie w RAM, jak `sysfs` i `procfs`, które nie mają fizycznego superbloku. W ich przypadku zawartość obiektu superbloku jest całkowicie wygenerowana. Typem obiektu superbloku jest `struct super_block`. Jest to struktura, która ma pola przechowujące takie dane jak: identyfikator urządzenia, na którym system plików jest zainstalowany, maksymalny rozmiar pliku w tym systemie, identyfikator typu system plików, liczba aktywnych odwołań do tego systemu, itd. Jednym z najważniejszych pól jest `s_op`, które wskazuje na obiekt operacji superbloku, nazywany także *tablicą metod superbloku*. Ten obiekt jest w rzeczywistości strukturą typu `struct super_operations`. Każde jej pole wskazuje na funkcję wywoływaną wtedy, gdy jądro musi wykonać określoną operację na superbloku.

Obiekt superbloku

Przykładowo, zmniejszenie liczby odwołań o 1 jest dokonywane poprzez wywołanie funkcji `put_super()` w następujący sposób:

```
sb->s_op->put_super(sb);
```

Zmienna `sb` jest wskaźnikiem do obiektu superbloku. Ponieważ język C, w przeciwieństwie do C++, nie posiada wskaźnika `this`, zmienna `sb` musi zostać przekazana do funkcji `put_super()`, aby ta wiedziała, na rzecz którego obiektu została wywołana. Do innych metod superbloku należą: `alloc_inode()` — tworzy i inicjuje obiekt i-węzła, `destroy_inode()` — usuwa obiekt i-węzła, `read_inode()` — odczytuje zawartość bloku i-węzła z nośnika i umieszcza ją w obiekcie i-węzła, `dirty_inode()` — metoda oznacza obiekt i-węzła jako zmodyfikowany; jego zawartość może różnić się od zawartości bloku i-węzła na nośniku, który reprezentuje, `write_inode()` — zapisuje dane z obiektu i-węzła do bloku i-węzła znajdującego się na nośniku, `drop_inode()` — metoda ta jest wywoływana, gdy zostanie wyzerowany licznik odwołań do obiektu i-węzła, w systemach uniksowych skutkuje to usunięciem i-węzła,

Obiekt superbloku

`evict_inode()` — metoda ta usuwa blok i-węzła, `put_super()` — ta metoda jest wywoływana, kiedy system plików jest odmontowywany, aby zmniejszyć o 1 licznik odwołań do obiektu superbloku; jeśli jego wartość spadnie do 0, to ta funkcja także usunie ten obiekt, `sync_fs()` — ta metoda zapisuje dane z obiektu superbloku, do superbloku na nośniku; innymi słowy aktualizuje superblok, `statfs()` — metoda zwracająca statystyki dotyczące systemu plików, `remount_fs()` — metoda montuje system plików z nowymi opcjami, `umount_begin()` — ta metoda przerywa operację montowania systemu plików; wykorzystywana przez sieciowe systemy plików, takie jak NFS. Nie wszystkie metody z tej listy wykonują operacje na superbloku. Część z nich operuje na i-węzłach. Również nie wszystkie z nich muszą być zaimplementowane w kodzie, który obsługuje rzeczywisty system plików. Wartości wskaźników na niezaimplementowane metody wynoszą `NULL`. Obiekt superbloku jest tworzony i inicjowany przez funkcję `alloc_super()`.

Obiekt i-węzła

Obiekty i-węzłów przechowują dane wymagane do przeprowadzenia operacji na plikach i katalogach reprezentowanych przez te obiekty. Dane te obejmują: identyfikator właściciela pliku, tzw. rzeczywisty numer urządzenia przechowującego plik, prawa dostępu do pliku, rozmiar pliku i identyfikator i-węzła. W systemach plików kompatybilnych z Uniksem obiekty i-węzłów reprezentują bloki i-węzłów, ale w przypadku innych systemów dane dla tych obiektów są pobierane bezpośrednio z plików lub innych miejsc na nośniku. Są również systemy plików, które nie mają wszystkich danych wymaganych przez obiekty i-węzłów. Wtedy używane są wartości domyślne. I-węzły mogą być powiązane nie tylko ze zwykłymi plikami, ale również ze specjalnymi, takimi jak *pliki urządzeń* i kolejki FIFO. W każdym obiekcie i-węzła znajdują się dwa pola, które wskazują na tablice metod. Pierwsze z nich nazywa się `f_op` i wskazuje na obiekt operacji na pliku, a drugie ma nazwę `i_op` i wskazuje na obiekt operacji na i-węźle.

Obiekt i-węzła

Operacje na i-węźle obejmują: `create()` — tworzy nowy obiekt i--węzła, `lookup()` — przeszukuje katalog w poszukiwaniu i-węzła związanego z określoną pozycją katalogu, `link()` — tworzy twarde dowiązanie, `unlink()` — usuwa dowiązanie, `symlink()` — tworzy dowiązanie symboliczne, `mkdir()` — tworzy katalog, `rmdir()` — usuwa pusty katalog, `mknod()` — tworzy plik specjalny, `rename()` — zmienia nazwę pliku, `readlink()` — kopiuje określoną część pełnej ścieżki związanej z określonym dowiązaniem, `follow_link()` — konwertuje dowiązanie symboliczne na wskazywany przez nie i-węzeł, `permissions()` — obsługuje prawa dostępu w niektórych systemach plików, `setattr()` — inicjuje zdarzenie powiadamiające o zmianie zawartości bloku i-węzła, `getattr()` — powiadamia, że obiekt i-węzła powinien być zaktualizowany danymi z bloku i-węzła, `setxattr()` — ustawia rozszerzone atrybuty, `getxattr()` — odczytuje wartość określonego atrybutu rozszerzonego, `listxattr()` — kopiuje do bufora listę rozszerzonych atrybutów, `removexattr()` — usuwa określony atrybut rozszerzony.

Obiekt wpisu katalogowego

Obiekty dentry są związane z każdą nazwą, która pojawia się w ścieżce. Przykładowo, jądro utworzy trzy takie obiekty dla następującej ścieżki: `/usr/java`. Pierwszy będzie związany ze znakiem `/` symbolizującym główny katalog (ang. *root*), drugi będzie reprezentował katalog `usr`, a ostatni będzie powiązany z katalogiem `java`. Obiekty dentry reprezentują również nazwy plików znajdujące się na końcach niektórych ścieżek i punkty montowania, które mogą występować w ścieżkach. Te obiekty nie mają swoich odpowiedników na nośniku. Są one tworzone na bieżąco, podczas analizy ścieżek i niezbędne do przeprowadzania operacji specyficznych dla katalogów, takich jak poruszanie się po drzewie katalogów. Obiekty dentry są reprezentowane przez struktury typu `struct dentry`. Każdy taki obiekt może znajdować się w jednym z trzech stanów: *używany*, *nieużywany* i *ujemny*. Obiekt dentry w stanie *używany* jest związany z prawidłowym obiektem i-węzła i był ostatnio używany przez jądro. Obiekt dentry w stanie *nieużywany* jest także związany z prawidłowym obiektem i-węzła, ale przez dłuższy czas nie był używany.

Obiekt wpisu katalogowego

Jądro nie usuwa takiego obiektu, chyba, że brakuje wolnej pamięci. Utrzymuje takie obiekty dentry, bo mogą być przydatne w przyszłości. Obiekt wpisu katalogowego w stanie *ujemny* nie jest powiązany z prawidłowym obiektem i-węzła. Oznacza to, że związany jest z plikiem lub katalogiem, który został usunięty lub nigdy nie istniał. Jądro także nie usuwa takiego obiektu bez powodu. Te obiekty mogą być użyteczne, jeśli ścieżka zawierająca pozycje z nimi związane będzie analizowana. Wtedy mogą zapobiec przetwarzaniu przez jądro nieprawidłowych ścieżek, które już były analizowane. Obiekty wpisów katalogowych są tworzone i usuwane przez alokator plastrowy. Jądro Linuksa utrzymuje także bufor obiektów dentry. Składa się on z trzech elementów: listy wszystkich obiektów dentry, listy ostatnio wykorzystywanych obiektów, która przechowuje zazwyczaj obiekty dentry w stanie *używany* i *nieużywany* i tablicy mieszającej (ang. *hash array*), która używa funkcji skrótu (ang. *hash function*) celem szybkiej lokalizacji określonego obiektu w buforze. Jądro rozpoczyna analizę ścieżki od jej ostatniej pozycji.

Obiekt wpisu katalogowego

Jeśli uda się odnaleźć obiekt dentry powiązany z tą nazwą w buforze, to istnieje szansa, że reszta obiektów dentry związanych z analizowaną ścieżką została już utworzona i jądro nie musi tworzyć ich na nowo. Istnieje również bufor obiektów i-węzłów powiązanych ze zbuforowanymi obiektami dentry. Tablica metody obiektów dentry jest reprezentowana przez strukturę typu `struct dentry_operations`. Do tych metod zaliczają się: `d_revalidate()` — ta metoda sprawdza poprawność obiektu dentry, `d_hash()` — jest to funkcja skrótu, `d_compare()` — porównuje ze sobą nazwy dwóch plików lub katalogów, `d_delete()` — ta metoda jest wywoływana, gdy wartość licznika odwołań do obiektu dentry spadnie do 0, `d_release()` — usuwa obiekt dentry, `d_iput()` — ta metoda jest wywoływana gdy obiekt dentry utraci obiekt i-węzła, z którym był powiązany.

Obiekt pliku

Dla procesu użytkownika najważniejszymi obiektami VFS są obiekty plików. Są one tworzone przez wywołanie systemowe `open()`, a usuwane przez `close()`. Obiekt pliku wskazuje na obiekt dentry, który z kolei wskazuje na obiekt i-węzła związany z plikiem otwartym przez proces użytkownika. Z pojedynczym plikiem może być powiązanych wiele obiektów plików, zależnie od tego ile razy został on otwarty przez procesy użytkownika. Typ obiektu pliku jest określony strukturą typu `struct file`, której jedno z pól wskazuje na strukturę typu `struct file_operations`. Ta ostatnia implementuje tablicę metod obiektu pliku, do których należą: `llseek()` — aktualizuje wskaźnik pliku, `read()` — odczytuje plik, `write()` — zapisuje plik, `poll()` — usypia proces użytkownika i budzi go, gdy zmienia się zawartość pliku, `unlocked_ioctl()` i `compat_ioctl()` — te dwie metody są używane do przeprowadzania operacji na pliku urządzenia, które nie mogą być zaimplementowane z użyciem zwykłych operacji plikowych; w starszych wersjach jądra była tylko jedna taka metoda, nazwana `ioctl()`, która używała blokady BKL;

Obiekt pliku

metody `unlocked_ioctl()` i `compat_ioctl()` nie używają jej, a ta druga dodatkowo zachowuje kompatybilność w obsłudze plików między 32- i 64-bitowymi platformami sprzętowymi; innymi słowy pozwala na wykonywanie 32-bitowych operacji plikowych na 64-bitowym sprzęcie, `mmap()` — odwzorowuje plik w RAM, `open()` — otwiera plik, `flush()` — zachowanie tej metody zależy od systemu plików, ale zawsze dekrementuje licznik odwołań do obiektu pliku, `release()` — jest wywoływana gdy wartość licznika odwołań obiektu pliku spadnie do 0, `fsync()` — zapisuje wszystkie zbuforowane zmiany na nośnik, `aio_fsync()` — robi to samo, co `fsync()`, ale bez usypiania procesu, który zapoczątkował tę operację, `fasync()` — aktywuje lub dezaktywuje sygnały powiadamiające o stanie operacji asynchronicznych, `read_iter()` — odczytuje dane z pliku i składuje je w wielu buforach, `write_iter()` — zapisuje dane z wielu buforów do pojedynczego pliku, `sendpage()` — przesyła dane między plikami, `get_unmapped_area()` — odwzorowuje plik na nieużywany obszar RAM, `lock()` — metoda ta zarządza blokadą pliku,

Obiekt pliku

`flock()` — metoda ta jest używana do implementacji wywołania systemowego, o tej samej nazwie, które realizuje doradcze blokowanie plików, `check_flags()` — sprawdza flagi ustawione przez funkcję `fcntl()`.

Inne struktury danych

Oprócz opisanych czterech typów obiektów, VFS używa kilku innych struktur danych. Struktury typu `file_system_type` przechowują dane o *typach systemów plików* obsługiwanych przez Linuksa. Te struktury są używane przez funkcję `get_sb()`, która odczytuje z nośnika zawartość superbloku danego systemu plików. Dla każdego obsługiwanego systemu plików jądro Linuksa ma po jednej takiej strukturze. Kiedy system plików jest montowany, jądro tworzy strukturę typu `struct vfsmount`, która przechowuje dane o punkcie montowania, w tym flagi określające jakie operacje mogą być przeprowadzane na tym systemie plików. Z każdym procesem użytkownika powiązane są trzy struktury danych VFS. Struktura typu `struct files_struct` przechowuje dane o plikach otwartych przez proces użytkownika i ich deskryptorach, w tym wskaźniki do obiektów plików. Struktura typu `fs_struct` przechowuje dane o systemie plików związanym z danym procesem, w tym nazwy katalogu bieżącego i głównego. Struktura typu `struct mnt_namespace` określa unikatowy dla procesu widok zamontowanego systemu plików.

Inne struktury danych

Dwie pierwsze z trzech opisanych struktur danych VFS związanych z procesami mogą być współdzielone przez spokrewnione procesy. Ostatnia jest domyślnie współdzielona przez wszystkie procesy w systemie, ale można także ją osobno zdefiniować dla określonego procesu.

Pytania

?

Koniec

Dziękuję Państwu za uwagę!