Zadanie 5

Monitorowanie zdarzeń VFS

Celem zadania jest przygotowanie implementacji mechanizmu monitorowania zdarzeń zachodzących w systemach plików. Mechanizm ten będzie udostępniał nowe wywołanie systemowe `VFS_NOTIFY`, które jest obsługiwane przez serwer *vfs* i które wstrzymuje wywołujący je proces, aż zajdzie zdarzenie określone przez parametry tego wywołania.

Uwaga: W poniższym opisie, jeśli nie zaznaczono inaczej, słowo *plik* jest używane w znaczeniu każdego obiektu, który w MINIX-ie implementuje interfejs pliku, m.in. zwykłego pliku, katalogu, dowiązania, pseudourządzenia itp.

VFS

VFS (ang. *Virtual File System*, pol. *Wirtualny System Plików*) to podsystem systemu operacyjnego umożliwiający jednolity dostęp do plików umieszczonych na różnych systemach plików. Jest on warstwą pośredniczącą między aplikacjami a podsystemami implementującymi konkretne systemy plików (MFS, ext2, procfs itd.). Przetwarza wywołania systemowe realizujące operacje na plikach, implementuje akcje wspólne dla różnych systemów plików oraz przekazuje żądania do odpowiednich systemów plików. Zarządza on także wszystkimi używanymi w systemie plikami i wszystkimi zamontowanymi systemami plików.

W MINIX-ie wirtualny system plików jest zaimplementowany jako serwer *vfs*. Więcej o jego budowie i sposobie działania można przeczytać na Wiki MINIX-a: VFS internals.

Wywołanie systemowe `VFS_NOTIFY`

Mechanizm monitorowania zdarzeń opiera się na nowym wywołaniu systemowym `VFS_NOTIFY` obsługiwanym przez serwer *vfs*. Argumentami tego wywołania są deskryptor pliku, który proces chce monitorować, oraz flaga oznaczająca rodzaj zdarzenia, o którym proces chce zostać powiadomiony. Proces jest wstrzymywany przez serwer *vfs* na wywołaniu tego wywoływania systemowego, aż zajdzie określone jego argumentami zdarzenie. Obsługiwane są następujące typy zdarzeń:

1. `NOTIFY_OPEN` – proces zostaje wstrzymany, aż plik wskazywany przez podany deskryptor zostanie otwarty. Proces jest wznawiany przez pierwsze otwarcie pliku, które nastąpi po wywołaniu `VFS_NOTIFY` i zakończy się sukcesem.

2. `NOTIFY_TRIOPEN`: proces zostaje wstrzymany, aż plik wskazywany przez podany deskryptor będzie **jednocześnie** otwarty trzy lub więcej razy. Jeśli wskazany plik będzie jednocześnie otwarty trzy lub więcej razu już w momencie wywoływania `VFS_NOTIFY`, to wywołanie powinno zakończyć się od razu (tj. proces nie jest wstrzymywany). Jeśli w momencie wywołania `VFS_NOTIFY` wskazany plik jest otwarty jednocześnie mniej niż trzy razy, to proces jest wstrzymywany aż do otwarcia, po którym ten plik będzie jednocześnie otwarty trzy razy (niezależnie od tego, które z kolei będzie to otwarcie).

3. `NOTIFY_CREATE`: proces zostaje wstrzymany, aż w katalogu wskazanym przez podany deskryptor zostanie utworzony nowy plik. Monitorowane są tylko utworzenia plików bezpośrednio w tym katalogu, nie są monitorowane utworzenia plików w podkatalogach monitorowanego katalogu.

4. `NOTIFY_MOVE`: proces zostaje wstrzymany, aż do katalogu wskazanego przez podany deskryptor zostanie przeniesiony plik z innego katalogu. Monitorowane są tylko przeniesienia plików bezpośrednio do tego katalogu, nie są monitorowane przeniesienia plików do podkatalogów monitorowanego katalogu. Nie są

monitorowane także przeniesienia w obrębie tego samego katalogu, czyli zmiany nazw plików bez zmiany ich położenia.

Mechanizm monitorowania zdarzeń działa według następującej specyfikacji:

1. Wstrzymany proces jest wznawiany przez pierwsze po wywołaniu `VFS_NOTIFY` wystąpienie monitorowanego zdarzenia, które zakończy się sukcesem. Proces nie jest wznawiany przez zdarzenia, które zakończyły się błędem.

2. Gdy proces zostanie wznowiony przez wystąpienie monitorowanego zdarzenia, wywołanie systemowe `VFS_NOTIFY` powinno zakończyć się statusem `OK`. Jeśli wywołanie systemowe `VFS_NOTIFY` nie może zostać zrealizowane, wywołanie powinno zakończyć się odpowiednim błędem. Na przykład `EBADF` – jeśli podany w wywołaniu deskryptor jest nieprawidłowy, `EINVAL` – jeśli podana w wywołaniu flaga oznaczająca typ monitorowanego zdarzenia jest nieprawidłowa, `ENOTDIR` – jeśli podany w wywołaniu deskryptor powinien wskazywać na katalog, a nie wskazuje itd.

3. Dane zdarzenie może być jednocześnie monitorowane przez wiele procesów. W momencie wystąpienia tego zdarzenia wznowione powinny zostać wszystkie procesy monitorujące to zdarzenie.

4. Jednocześnie monitorowane jest co najwyżej `NR_NOTIFY` (stała zdefiniowana w załączniku do zadania) zdarzeń. Innymi słowy, co najwyżej `NR_NOTIFY` procesów może być jednocześnie wstrzymanych na wywołaniu `VFS_NOTIFY`. Wywołania, których obsługa oznaczałaby przekroczenie tego limitu, powinny zakończyć się błędem `ENONOTIFY`.

Załączniki do zadania

Do zadania załączona jest łatka `zadanie5-szablon.patch`, która definiuje stałe opisane w zadaniu i dodaje wywołanie systemowe `VFS_NOTIFY` realizowane przez funkcję `int do_notify(void)` dodaną do serwera *vfs*.

Do zadania załączony jest także przykładowy program `test-notify-open.c`, który ilustruje użycie wywołania systemowego `VFS_NOTIFY` do monitorowania zdarzenia typu `NOTIFY_OPEN` pliku podanego jako argument programu. Przykładowy scenariusz użycia programu:

```
# make test-notify-open
clang -O2   -o test-notify-open test-notify-open.c
# touch ./test.txt
# ./test-notify-open ./test.txt
```

Tak uruchomiony program `test-notify-open` zakończy się dopiero, gdy plik `test.txt` zostanie otwarty przez inny proces, np. przez wykonanie w innym terminalu polecenia:

```
# cat ./test.txt
```

Wtedy program `test-notify-open` zakończy się, wypisując:

```
Wynik VFS_NOTIFY: 0, errno: 0
```

Wymagania i niewymagania

1. Wszystkie pozostałe operacje realizowane przez serwer *vfs*, inne niż opisane powyżej, powinny działać bez zmian.

2. Modyfikacje serwera *vfs* nie mogą powodować błędów w systemie (m.in. *kernel panic*) i błędów w systemie plików (m.in. niespójności danych).

3. Modyfikacje mogą dotyczyć tylko serwera *vfs* (czyli mogą dotyczyć tylko plików w katalogu `/usr/src/minix/servers/vfs/`), za wyjątkiem modyfikacji dodawanych przez łatkę `zadanie5-szablon.patch`. Nie można zmieniać definicji dodawanych przez tę łatkę.

4. Podczas działania zmodyfikowany serwer nie może wypisywać żadnych dodatkowych informacji na konsolę ani do rejestru zdarzeń (ang. *log*).

5. Rozwiązanie będzie testowane z plikami będącymi zwykłymi plikami, katalogami, dowiązaniami (twardymi i miękkimi), pseudourządzeniami (np. `/dev/null`) i pseudoplikami systemu plików *procfs* (np. `/proc/cpuinfo`). Można założyć, że rozwiązanie nie będzie testowane z innymi rodzajami plików, np. z potokami (ang. *pipe*, *fifo*).

6. Rozwiązanie nie musi być optymalne pod względem prędkości działania. Akceptowane będą rozwiązania, które działają bez zauważalnej dla użytkownika zwłoki.

Wskazówki

1. Implementacja serwera *vfs* nie jest omawiana na zajęciach, ponieważ jednym z celów tego zadania jest samodzielne przeanalizowanie odpowiednich fragmentów kodu źródłowego MINIX-a. Rozwiązując to zadanie, należy więcej kodu przeczytać, niż samodzielnie napisać lub zmodyfikować.

2. Przykład, jak w serwerze *vfs* zaimplementowane jest wstrzymywanie procesów, można znaleźć, analizując działanie wywołania systemowego `VFS_OPEN` dla potoków (ang. *pipe*, *fifo*), w tym funkcję `pipe_open()`. Inny przykład można znaleźć, analizując implementację blokad plików (ang. *file lock*, *flock*) realizowaną m.in. przez wywołanie systemowe `VFS_FCNTL` i funkcję `lock_op()`.

3. W Internecie można znaleźć wiele opracowań omawiających działanie serwera *vfs*, np. slajdy do wykładu Prashant Shenoy'a. Korzystając z takich materiałów, należy jednak zwrócić uwagę, czy dotyczą one tej samej wersji MINIX-a i serwera *vfs*, którą modyfikujemy w tym zadaniu.

Rozwiązanie

Poniżej przyjmujemy, że *ab123456* oznacza identyfikator studenta rozwiązującego zadanie. Należy przygotować łatkę (ang. *patch*) ze zmianami w sposób opisany w treści zadania 3. Rozwiązanie w postaci łatki `ab123456.patch` należy umieścić w Moodle. Łatka będzie aplikowana do nowej kopii MINIX-a. **Uwaga:** łatka z rozwiązaniem powinna zawierać także zmiany dodane przez łatkę `zadanie5-szablon.patch`. Należy zadbać, aby łatka zawierała tylko niezbędne różnice.

W celu skompilowania i uruchomienia systemu ze zmodyfikowanym serwerem *vfs* wykonane będą następujące polecenia:

```
cd /
patch -t -p1 < ab123456.patch
cd /usr/src; make includes
cd /usr/src/minix/servers/vfs/; make clean && make && make install
cd /usr/src/releasetools; make do-hdboot
reboot
```

Ocenianie

Oceniana będą zarówno poprawność, jak i styl rozwiązania. Podstawą do oceny rozwiązania będą testy automatyczne sprawdzające poprawność implementacji oraz przejrzenie kodu przez sprawdzającego. Rozwiązanie, w którym łatka nie nakłada się poprawnie, które nie kompiluje się lub powoduje *kernel panic* podczas uruchamiania, otrzyma 0 pkt.

Wymaganą częścią zadania jest implementacja mechanizmu monitorowania zdarzeń typu `NOTIFY_OPEN`. Za poprawną i w dobrym stylu pełną implementację tej funkcjonalności rozwiązanie otrzyma 3 pkt. Ponadto rozwiązanie może implementować monitorowanie dowolnego podzbioru pozostałych typów zdarzeń. Za poprawną i w dobrym stylu ich implementację rozwiązanie otrzyma: `NOTIFY_TRIOPEN` – 0,5 pkt., `NOTIFY_CREATE` – 1 pkt, `NOTIFY_MOVE` – 0,5 pkt. Jeśli monitorowanie któregoś typu nie jest obsługiwane przez rozwiązanie, wywołanie `VFS_NOTIFY` z oznaczającą je flagą powinno zakończyć się błędem `EINVAL`.