Systemy Operacyjne 2

Wywołania Systemowe

Arkadiusz Chrobot

Katedra Systemów Informatycznych

2 kwietnia 2024

Plan

Wprowadzenie

Wywołania systemowe są funkcjami jądra, które mogą być wywoływane z przestrzeni użytkownika. Tworzą one interfejs między oprogramowaniem użytkowym, a systemem operacyjnym. Ze względów bezpieczeństwa (ang. *safety*) współczesne systemy operacyjne nie pozwalają procesom użytkownika na bezpośrednią interakcję z zasobami, np. sprzętowymi, a nawet między nimi samymi (pamięć dzielona jest tutaj pewnym wyjątkiem). Niemniej jednak, aby wykonywać swoje zadania procesy potrzebują zasobów i mogą uzyskać do nich pośredni dostęp przy użyciu usług dostarczanych przez system operacyjny poprzez wywołania systemowe. To rozwiązanie ma kilka zalet. Wywołania systemowe upraszczają użycie zasobów, szczególnie tych sprzętowych, przez oprogramowanie użytkowe. Programiści, którzy tworzą takie programy nie muszą znać szczegółów użycia takiego sprzętu, po prostu polegają w tym zakresie na systemie operacyjnym. Dodatkowo, wywołania systemowe są niezbędne do zrealizowania *wirtualizacji* zasobów, takich jak CPU lub pamięć.

Application Programming Interface — API

Procesy użytkownika zazwyczaj nie wywołują bezpośrednio wywołań systemowych, ale robią to za pomocą interfejsu programowania aplikacji (ang. *Application Programming Interface*), nazywanego krótko API. Jest to zbiór funkcji, zmiennych i innych elementów oprogramowania, które ułatwiają procesom wykonywanie najważniejszych operacji i pozwalają na współpracę z systemem operacyjnym. API Linuksa jest zapożyczone z systemu Unix i zgodne ze standardami POSIX oraz SUS. Jest ono w większości zaimplementowane w *standardowej bibliotece języka C*, nazywanej w Linuksie *glibc* (*libc* w Uniksie). Niektóre z funkcji w niej dostępnych nie używają żadnych wywołań systemowych (np. `strcpy()`), inne są prostymi opakowaniami na wywołania systemowe (ang. *wrappings*), jak np. funkcja `write()`, a inne oprócz uruchamiania wywołania systemowego wykonują jeszcze dodatkowe operacje (jak np. funkcja `printf()`).

Wywoływanie wywołań systemowych

Proces użytkownika nie może wywołać wywołania systemowego jak zwykłej funkcji bo jest ono częścią przestrzeni jądra. Współczesne procesory oferują specjalny rozkaz ogólnie nazwany *przerwaniem programowym*, [1] który może być wykonany przez oprogramowanie użytkowe i spowodować przełączenie CPU w tryb systemowy i przekazanie sterowania do jądra, tak jak wtedy gdy obsługiwane jest przerwanie sprzętowe lub wyjątek. Taki rozkaz ma różne mnemoniki, w zależności od procesora. W przypadku procesorów PowerPC jest to `sc`. W 32-bitowych procesorach firm Intel i AMD nazywa się `int` i posiada operand (argument), którym jest numer przerwania (w przypadku Linuksa `0x80` — 128 dziesiętnie). Nowsze 32-bitowe procesor Intela i pokrewne, począwszy od modelu Pentium II, posiadają rozkaz `sysenter`. Procesory 64-bitowe firm AMD and Intel mają instrukcję `syscall`.

[1] To określenie jest wieloznaczne w terminologii Linuksa, więc trzeba zachować ostrożność posługując się nim.

Wywoływanie wywołań systemowych

32-bitowe procesory x86

Przerwanie programowe jest obsługiwane przez funkcję jądra o nazwie `system_call()`. Jej implementacja i działanie zależą od budowy procesora, na którym jest wykonywane jądro. Dla 32-bitowych procesorów x86 jest ona zdefiniowana w pliku `entry_32.S` (rozszerzenie `.S` oznacza, że plik zawiera kod zapisany w asemblerze). Funkcja `system_call()` jest również bramką wejściową (ang. *call gate*) dla wszystkich wywołań systemowych. Po wywołaniu najpierw sprawdza ona zawartość rejestru `eax`. Powinien on zawierać *numer wywołania systemowego*, który jest unikatowy dla każdego wywołania systemowego i jest również indeksem w tablicy `sys_call_table`. Ta tablica zawiera adresy wszystkich wywołań systemowych i jest zdefiniowana w pliku `syscall_table_32.S`, który jest włączony do pliku `entry_32.S`. To oznacza, że jest ona również zaimplementowana w asemblerze.

Wywoływanie wywołań systemowych

32-bitowe procesory x86

Jeśli numer w rejestrze `eax` jest nieprawidłowy, większy niż powinien, funkcja `system_call()` zwraca wartość `-ENOSYS`, informując, że proces starał się wywołać nieistniejące wywołanie systemowe. W przeciwnym przypadku funkcja `system_call()` odkłada na stos jądra procesu zawartość rejestrów `ebx`, `ecx`, `edx`, `edi` i `esi`. Te rejestry mogą przechowywać argumenty dla wywołania systemowego. Następnie wspomniana funkcja mnoży numer wywołania systemowego przez 4 (rozmiar 32-bitowego adresu w bajtach), a wyniku używa jako przesunięcia względem początku tablicy `sys_call_table`, by odnaleźć w niej adres wywołania systemowego. Potem funkcja `system_call()` używa tego adresu i zwykłego rozkazu `call`, aby uruchomić to wywołanie systemowe. Kończąc działanie to wywołanie zwraca wynik zapisując go do rejestru `eax` i oddaje sterowanie z powrotem do funkcji `system_call()`.

Wywoływanie wywołań systemowych

64-bitowe procesory x86

W przypadku 64-bitowych procesorów produkowanych przez firmy Intel i AMD funkcja `system_call()` jest zdefiniowana w pliku o nazwie `entry_64.S`, a tablica `sys_call_table` w pliku `unistd_64.h`, który jest włączany do `syscall_64.c`, a ten z kolei jest włączany do pliku `entry_64.S`. Numer wywołania systemowego jest zapisywany w rejestrze `rax`, w którym również to wywołanie zapisuje swój wynik. Wspomniany numer jest mnożony przez 8 (rozmiar 64-bitowego adresu w bajtach), zamiast 4. Argumenty dla wywołań systemowych mogą być przechowywane w rejestrach `rdi`, `rsi`, `rdx`, `r10`, `r9` i `r8`, ale ich wartości **nie są odkładane** na stos jądra procesu.

Tworzenie wywołań systemowych

Nagłówek wywołania systemowego

Wywołania systemowe są zwykłymi funkcjami jądra wykonywanymi w kontekście procesu, ale są one definiowane w szczególny sposób. Nazwy tych funkcji zaczynają się przedrostkiem `sys_`. Przykładowo, nazwa funkcji implementującej wywołanie systemowe `write()` to `sys_write()`. Każde wywołanie systemowe zwraca liczbę typu `long`, która zazwyczaj, ale nie zawsze, informuje o powodzeniu lub wyjątku wykonania. W takim przypadku 0 oznacza sukces, a liczba ujemna wskazuje na przyczynę problemu. Dla niektórych wywołań systemowych znaczenie tych liczb może być jednak inne. W przypadku 32-bitowych procesorów x86 wywołanie może mieć do 5 argumentów, a w przypadku 64-bitowych CPU o tej samej architekturze, do 6. Te argumenty są wstępnie przechowywane w rejestrach.

Tworzenie wywołań systemowych

Nagłówek wywołania systemowego

Nagłówek wywołania systemowego zawiera makro `asmlinkage`, które informuje kompilator, że argumenty dla tej funkcji są przekazywane za pośrednictwem stosu jądra procesu. Jeśli CPU tego nie wymaga (jak w przypadku 64-bitowych procesorów x86), to wspomniane makro jest puste. Aby uprościć tworzenie nagłówków wywołań systemowych, programiści jądra Linuksa dodali makra o nazwach `SYSCALL_DEFINE`n, gdzie n jest liczbą parametrów wymaganych przez to wywołanie i zawiera się w przedziale od 0 do 5 lub 6 w przypadku procesorów x86. Makro `SYSCALL_DEFINE0` jest używane do budowy nagłówka wywołania systemowego, które nie potrzebuje parametrów. Jeśli wywołanie potrzebuje 2 parametrów, to używane jest makro `SYSCALL_DEFINE2`. Jednakże, każde z tych makr wymaga $2 \cdot n + 1$ argumentów. Pierwszym jest nazwa wywołania systemowego (bez przedrostka `sys_`). Po niej, jeśli są potrzebne, następują pary argumentów złożone z nazwy i typu parametru wywołania systemowego.

Tworzenie wywołań systemowych

Ciało wywołania systemowego — ogólne uwagi

Prawie każde wywołanie systemowe ma efekty uboczne, czyli zmienia stan wewnętrzny jądra systemu operacyjnego. Ponieważ są one wykonywane w kontekście procesu, to mogą uzyskać dostęp do deskryptora procesu, który je wywołał, przy pomocy makra `current` i zmienić stan procesu na jeden ze stanów oczekiwania. W terminologii Linuksa nazywa się taką zmianę „uśpieniem procesu". Jeśli wywołanie systemowe wymaga więcej niż 5 lub 6 argumentów, albo nie mieszczą się one w rejestrach, to ich wartości mogą zostać umieszczone w określonym obszarze pamięci procesu, a adres początku tego obszaru może zostać przekazany do wywołania systemowego, przez któryś z tych rejestrów. Każde wywołanie systemowe, które otrzymuje argumenty musi je zweryfikować, zanim ich użyje. Danym pochodzącym z przestrzeni użytkownika jądro nie może ufać. Szczególnie dobrze należy sprawdzić argumenty, które są wskaźnikami (adresami w pamięci).

Tworzenie wywołań systemowych

Ciało wywołania systemowego — ogólne uwagi

Weryfikacja wskaźników obejmuje trzy etapy:

1. Czy wskaźnik wskazuje obszar pamięci w przestrzeni użytkownika?
2. Czy wskaźnik wskazuje obszar pamięci, który należy do procesu, który uruchomił wywołanie systemowe?
3. W zależności od operacji, która ma być przeprowadzona, wywołanie systemowe sprawdza, czy proces, który je uruchomił, ma uprawnienia do odczytu, zapisu lub wykonania powiązane z tym obszarem pamięci.

Wywołanie systemowe może także musieć sprawdzić uprawnienia wywołującego je procesu do innych zasobów. Począwszy do serii 2.6 jądra do tego celu służy funkcja `capable()`. Jest ona zdefiniowana w pliku nagłówkowym `linux/capability.h`, razem ze stałymi, które mogą być użyte jako jej argumenty i które określają różne uprawnienia. Jeśli proces ma dane uprawnienie, to funkcja zwróci wartość różną od 0, w przeciwnym przypadku 0.

Tworzenie wywołań systemowych

Ciało wywołania systemowego — ogólne uwagi

We wcześniejszych wersjach jądra wywołania systemowe sprawdzały tylko, czy wywołujący je proces należy do użytkownika uprzywilejowanego. W tym celu korzystały z funkcji `suser()`. Jeśli wywołanie wymaga skopiowania danych z przestrzeni użytkownika do przestrzeni jądra, to należy w tym celu użyć funkcji `copy_from_user()`. Jeśli kopiowanie ma przebiegać w odwrotnym kierunku, to należy skorzystać z funkcji `copy_to_user()`. Obie te funkcje przyjmują trzy argumenty: adres początkowy docelowego obszaru w pamięci, adres początkowy źródłowego obszaru w pamięci i liczbę bajtów do skopiowania. Również znaczenie wartości zwracanej jest w obu przypadkach takie samo. W przypadku sukcesu obie funkcje zwracają 0, a w przypadku niepowodzenia liczbę bajtów pozostałych do przekopiowania.

Tworzenie wywołań systemowych

Wywołanie systemowe `sys_ni_call()`

Istnieje specjalne wywołanie systemowe o nazwie `sys_ni_call()`. Zwraca ono wyłącznie wartość `-ENOSYS`. Ta funkcja została zdefiniowana dla wywołań systemowych, które nie zostały zaimplementowane dla danej platformy systemowej (np. są dostępne dla procesorów x86, ale nie PowerPC) lub dla takich, które z pewnych względów zostały usunięte. Na szczęście ta druga sytuacja nie zdarza się często w kodzie źródłowym jądra Linuksa.

Tworzenie wywołań systemowych

Wartość zwracana

Wartość zwrócona przez wywołanie systemowe zapisywana jest w rejestrze `eax` lub `rax` w przypadku procesorów x86. Stamtąd trafia ostatecznie do zmiennej `errno` dostępnej w przestrzeni użytkownika. Jeśli działanie wywołania systemowego zakończy się niepowodzeniem, to ta wartość może zostać użyta np. przez funkcję `perror()` do wypisania na ekranie czytelnego dla użytkownika opisu przyczyny tego niepowodzenia.

Dodawanie nowego wywołania systemowego

Dodanie nowego wywołania systemowego jest względnie proste. Przykładowo, dla 32-bitowych procesorów x86 wymaga ta czynność zdefiniowania takiego wywołania w jednym z plików źródłowych podsystemu jądra powiązanego z tym wywołaniem systemowym, dodania nowego wpisu w tablicy wywołań systemowych, umieszczonej w pliku `syscall_table_32.S`, określenia numeru tego wywołania w pliku nagłówkowym `include/asm/unistd_32.h` oraz skompilowania i instalacji jądra. Podobne kroki należy przeprowadzić dla 64-bitowych procesorów x86, ale dodanie nowej pozycji do tablicy wywołań systemowych i określenie numeru tego wywołania wymaga jedynie modyfikacji pliku nagłówkowego `include/asm/unistd_64.h`.
W przypadku nowszych wersji jądra dodanie nowego wywołania systemowego jest jeszcze ▸ prostsze. Wystarczy dodać tylko odpowiedni wpis w pliku `syscall_64.tbl` lub `syscall_32.tbl` oraz umieścić prototyp wywołania w pliku nagłówkowym `syscalls.h`.

Dodawanie nowego wywołania systemowego

Wywołanie nowego wywołania systemowego

Ponieważ żadna funkcja API nie zna nowego wywołania systemowego, to żadna go nie wywoła. Istnieje jednak możliwość jego wywołania z poziomu przestrzeni użytkownika. Zanim pojawiała się wersja jądra 2.6.18, do tego celu służyły makra o nazwie `_syscall`n, gdzie n określa liczbę argumentów pobieranych przez wywołanie systemowe. Każde z tych makr wymaga $2 \cdot n + 2$ własnych argumentów. Są nimi: typ wartości zwracanej przez wywołanie systemowe, nazwa tego wywołania i opcjonalnie para lub pary składające się z typu i wartości argumentu. Niestety, te makra nie były dostępne dla wszystkich platform sprzętowych obsługiwanych przez Linuksa, dlatego w wersji 2.6.18 jądra zostały zastąpione przez funkcję `syscall()`, która pobiera jeden obowiązkowy argument — numer wywołania systemowego i dowolną liczbę argumentów wymaganych przez to wywołanie. Ta funkcja zwraca tę samą wartość, jak uruchamiane przez nią wywołanie systemowe.

Dodawanie nowego wywołania systemowego

Zalety i wady dodawania nowego wywołania systemowego

Zalety:

1. Wywołania systemowe można względnie prosto opracować i używać.
2. Wywołania systemowe w Linuksie są bardzo wydajne.

Wady:

1. Nowemu wywołaniu systemowemu należy przydzielić unikatowy numer, który musi być zaakceptowany przez wszystkich programistów jądra Linuksa.
2. Interfejs tego wywołania (liczba i kolejność jego argumentów) nie może się w przyszłości zmienić.
3. Może zajść konieczność zdefiniowania go dla wszystkich platform sprzętowych obsługiwanych przez jądro Linuksa.
4. Wywołania systemowe nie powinno być środkiem komunikacji dla procesów użytkownika.
5. Definicja wywołania systemowego nie może być zawarta w module jądra.

Podsumowanie

Oryginalna wersja Uniksa posiadała około 100 wywołań systemowych. Linux ma ich około 400. Ta liczba jest różna w zależności od wspieranej przez jądro platformy sprzętowej, ale na ogół nie zmienia się często. Większość z tych wywołań zaimplementowana jest według zasady „rób jedną rzecz, ale rób ją dobrze”. Jednym z tych, w których budowie nie została ta zasada uwzględniona jest `ioctl()`. Należy unikać dodawania nowych wywołań systemowych. Z niektórymi z problemów, których rozwiązaniem początkowo może się wydawać dodanie nowego wywołania systemowego, można sobie poradzić z pomocą podsystemu obsługi urządzeń lub systemu plików.

Pytania

?

Koniec

Dziękuję Państwu za uwagę!