Systemy Operacyjne 2

Obsługa Przerwań

Arkadiusz Chrobot

Katedra Systemów Informatycznych

11 kwietnia 2024

Plan

Wstęp

Przerwania są istotnym elementem każdego systemu komputerowego. Są one używane do obsługi wyjątków i komunikacji z urządzeniami wejścia-wyjścia. Przykładem ich zastosowań są wywołania systemowe. Implementacja przerwań jest w dużym stopniu zależna od sprzętu. Na tym wykładzie mechanizm przerwań i ich obsługi w jądrze Linuksa zostanie przedstawiony w sposób ogólny. Bardziej zaawansowane zagadnienia, jak przerwania międzyprocesorowe (ang. *inter–processor interrupts*) lub równoważenie przerwań (ang. *interrupts balancing*) w systemach wieloprocesorowych, zostaną pominięte.

Przerwania

W jądrze Linuksa rozróżniane są dwie kategorie przerwań:

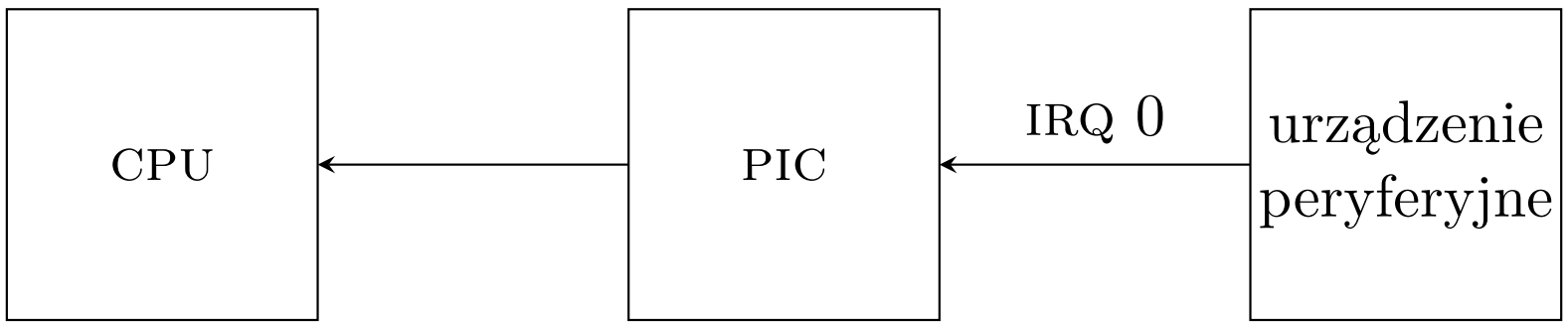
wyjątki Są to wysoko-priorytetowe przerwania powiązane z ważnymi zdarzeniami, takimi jak błąd strony, całkowitoliczbowe dzielenie przez zero, wywołanie systemowe, które wymagają natychmiastowej obsługi przez CPU i nie mogą być zignorowane. Wyjątki są zazwyczaj synchroniczne, co oznacza, że pojawiają się jako efekt uboczny wykonania określonego rozkazu procesora. Funkcje jądra odpowiedzialne za ich obsługę są wykonywane w kontekście procesu i jest dla nich użyteczna wartość zwracana przez makro `current`.

przerwania sprzętowe Są one używane przez urządzenia peryferyjne do powiadamiania, że wymagają obsługi przez CPU. Te przerwania są asynchroniczne, co oznacza, że mogą się pojawić w dowolnym momencie. Funkcje jądra obsługujące te przerwania muszą działać szybko, dlatego wykonywane są w *kontekście przerwania*.

Struktura sprzętowa

System przerwań wymaga wspomagania sprzętowego. Schemat takiego sprzętu w systemie uniprocesorowym przedstawia Rys. 1. Każde urządzenie peryferyjne jest połączone przy pomocy *linii zgłaszania przerwań* (ang. *Interrupt Request Line*), nazywanej linią IRQ z układem scalonym, który jest *programowalnym kontrolerem przerwań* (ang. *Programmable Interrupt Controller*), czyli PIC. Z każdą linią IRQ związany jest unikatowy numer (będący zazwyczaj liczbą naturalną), który identyfikuje źródło przerwania. Urządzenie peryferyjne zgłasza przerwania zmieniając stan linii IRQ. Układ PIC wykrywa tę zmianę, określa numer przerwania oraz powiadamia procesor, który wykonuje funkcję jądra nazywaną *procedurą obsługi przerwania* (ang. *interrupt service routine* — ISR). Opisywane rozwiązanie to *wektorowy system przerwań* (ang. *vectored interrupt*), który cechuje szybkie określenie źródła przerwania.

Struktura sprzętowa



Rysunek 1: Ogólna struktura sprzętowa dla przerwań

Struktura sprzętowa

Wektorowy system przerwań jest wydajny tylko wtedy, gdy każde urządzenie wyjścia-wejścia ma własna linię IRQ. Niestety, część współczesnych komputerów (w tym te bazujące na procesorach x86) ma ograniczoną liczbę takich linii, więc pozwalają urządzeniom na współdzielenie niektórych z nich. To oznacza, że wektorowy system przerwań jest połączony z techniką *odpytywania* (ang. *polling*). Za każdym razem, kiedy na współdzielonej linii zgłaszane jest przerwanie, jądro systemu musi sprawdzić, które z urządzeń do niej podłączonych jest źródłem tego zgłoszenia. To jest czynność zajmująca czas. Programiści jądra Linuksa musieli wziąć pod uwagę wszystkie różnice w budowie sprzętu związanego z przerwaniami, występujące między różnymi systemami komputerowymi, a niekiedy między kolejnymi wersjami tego samego systemu komputerowego. Przykładowo w komputerach bazujących na procesorach x86 podstawowy sterownik PIC został zastąpiony przez *zaawansowany programowalny kontroler przerwań* — APIC. Co więcej, w systemach wieloprocesorowych każdy procesor ma swój APIC nazywany LAPIC.

Obsługa przerwania

Aby poradzić sobie ze wspomnianymi różnicami programiści jądra Linuksa podzielili kod odpowiedzialny za obsługę przerwań na trzy warstwy:

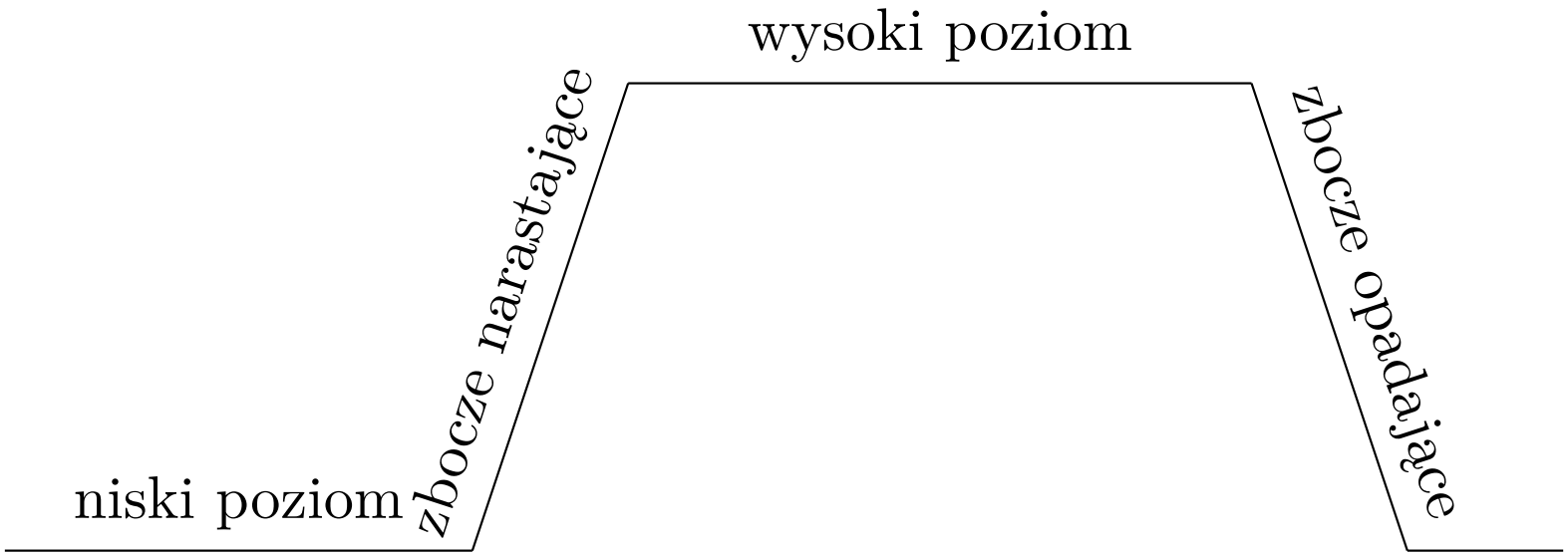
wysokopoziomowe procedury obsługi przerwań to zbiór funkcji jądra odpowiedzialnych za obsługę przerwań,

obsługa przepływu przerwania kod jądra zajmujący się różnicami w obsłudze przerwań zgłaszanych poziomem, zboczem (Rys. 2), przerwań przypisanych do konkretnego procesora i innych,

enkapsulacja niskopoziomowego sprzętu ta warstwa jest odpowiedzialna na obsługę różnorodnych układów PIC, w różnych systemach komputerowych, a czasami w obrębie tego samego systemu.

Wszystkie te warstwy są połączone za pomocą tablicy deskryptorów przerwań nazwanej `irq_desc`. Każdy jej element przechowuje wskaźniki do procedur obsługi przerwań i funkcji obsługujących układy PIC.

Zgłaszanie przerwania



Rysunek 2: Sygnał binarny

Przetwarzanie przerwania

Kiedy procesor otrzymuje sygnał przerwania, to automatycznie przełącza się w tryb systemowy (o ile już w nim nie był) i wykonuje zależny od jego architektury kod jądra, napisany w asemblerze, który odkłada na stos jądra procesu wartości rejestrów i przygotowuje środowisko do wykonania funkcji napisanych w języku C. Kod asemblerowy jest umieszczony w pliku `entry.S`, właściwym dla określonej platformy sprzętowej (w przypadku 32-bitowych procesorów x86 jest to plik `entry_32.S`, a dla 64-bitowych procesorów o tej samej architekturze — plik `entry_64.S`). Po wykonaniu tego kodu sterowanie, w przypadku większości platform sprzętowych, jest przekazywane do funkcji `do_IRQ()`. Wyjątkami są komputery z procesorami o architekturze Sparc, Sparc64 i Alpha, ale one nie będą omawiane na tym wykładzie.

Przetwarzanie przerwania

Funkcja `do_IRQ()`

Implementacja funkcji `do_IRQ()` również jest zależna od platformy sprzętowej, ale jej schemat działania jest zawsze ten sam. Najpierw zapisuje ona wartości rejestrów do struktury typu `struct pt_regs`, której budowa jest także zależna od platformy sprzętowej. Następnie funkcja `do_IRQ()` używa numeru przerwania, który jest zapisany w jednym z rejestrów, jako indeksu w tablicy `irq_desc` i blokuje linię związaną z tym przerwaniem za pomocą funkcji z warstwy enkapsulacji niskopoziomowego sprzętu, wskazywanych przez deskryptor przerwania. Przykładowo, w przypadku 32-bitowych procesorów x86, używa ona funkcji `mask_and_ack_8259A()`. W przypadku niektórych przerwań `do_IRQ()` musi zablokować cały system przerwań (lub przynajmniej lokalny system przerwań CPU obsługującego to przerwanie). Po zablokowaniu linii IRQ funkcja sprawdza, czy jest dla tego przerwania zarejestrowana procedura obsługi i jeśli tak, to ją uruchamia.

Przetwarzanie przerwania

Funkcja `do_IRQ()`

Jeśli dla danego przerwania jest zarejestrowana więcej niż jedna procedura obsługi, to `do_IRQ()` wywołuje je po kolei i oczekuje, że któraś z nich zwróci wartość `IRQ_HANDLED`, oznaczającą, że przerwanie zostało obsłużone. Jeśli żadna procedura obsługi nie została zarejestrowana dla przerwania, to `do_IRQ()` zwraca wartość sygnalizującą błąd. Po obsłużeniu przerwania wywoływana jest funkcja `add_interrupt_randomness()`, aby uzupełnić pulę entropii jądra nowymi wartościami (wyjaśnione dalej), odblokowywana jest linia IRQ lub włączany ponownie cały (lub lokalny) system przerwań i następuje powrót z funkcji `do_IRQ()`. Większość wymienionych operacji wspomniana funkcja nie wykonuje samodzielnie, ale z pomocą innych funkcji, takich jak `handle_irq_event()` i `ret_form_intr()`.

Przetwarzanie przerwania

Pula entropii

Pula entropii zawiera wartości, które służą do inicjacji (ang. *seed*) znajdujących się w jądrze, dwóch kryptograficznie bezpiecznych generatorów liczby pseudolosowych. Te generatory są dostępne dla przestrzeni użytkownika jako dwa urządzenia znakowe reprezentowane przez pliki `/dev/random` i `/dev/urandom`, które mogą być czytane tak, jak zwykłe pliki tekstowe. Pierwsze urządzenie wstrzymuje odczyt, gdy żądana ilość entropii nie jest dostępna, a drugie nigdy tego nie robi. Oba generują bezpieczne liczby pseudolosowe, ale w rzadkich przypadkach (zwykle, gdy takie liczby są potrzebne w trakcie inicjacji jądra), wskazane jest użycie generatora `/dev/random`. Współcześnie, w Linuksie wszystkie przerwania wnoszą wkład do puli entropii, ale nie bezpośrednio. W ramach obsługi przerwania jądro odczytuje kilka wartości z różnych zasobów, które są dobrymi źródłami losowości, jak np. niektóre rejestry CPU.

Rejestracja i wyrejestrowanie procedury obsługi przerwania

Dla programisty sterownika urządzenia najistotniejszymi funkcjami jądra związanymi z obsługą przerwań są te, które pozwalają na rejestrację i wyrejestrowanie procedur obsługi przerwań. Funkcja rejestrująca ma następujący prototyp:

```
int request_irq(unsigned int irq, irq_handler_t
handler, unsigned long irqflags, const char *name, void
*dev)
```

Zwraca ona 0 w przypadku sukcesu i wartość `-EBUSY` w przypadku niepowodzenia. Funkcja ta nie tylko rejestruje procedurę obsługi przerwania, ale także aktywuje linię IRQ z nim związaną. Wymaga ona pięciu argumentów. Pierwszym jest numer przerwania, drugim adres procedury obsługi przerwania, a trzecim jest flaga lub suma bitowa zgodnych flag. Flagi są zdefiniowane w pliku nagłówkowym `linux/interrupt.h`

Rejestracja i wyrejestrowanie procedury obsługi przerwania

Spośród flag związanych z rejestracją przerwań najbardziej interesujące są następujące: `IRQF_TIMER` — procedura obsługi przerwania będzie obsługiwała przerwanie zegarowe, `IRQF_PERCPU` — procedura obsługi przerwania będzie wykonywana tylko na określonym CPU w systemie wieloprocesorowym, `IRQF_ONESHOT` — linia IRQ nie jest reaktywowana natychmiast po zakończeniu procedury obsługi przerwania, `IRQF_SHARED` — procedura obsługi przerwania jest rejestrowana dla linii IRQ, która jest współdzielona przez kilka urządzeń i innych procedur obsługi przerwań.

Rejestracja i wyrejestrowanie procedury obsługi przerwania

Czwarty argument funkcji `request_iqr()` to ciąg znaków określający nazwę urządzenia będącego źródłem danego przerwania. Ta nazwa jest używana w pliku tekstowym `proc/interrupts`[1]. Zawiera on statystyki obsłużonych (lub nie) przerwań, takie jak numer linii IRQ, numer obsługującego CPU, typ przerwania, nazwę PIC, nazwę urządzenia i inne. Inne statystyki o przerwaniach znajdują się w katalogach `proc/irq`.

Ostatni argument może być wartością `NULL`, jeśli linia IRQ nie jest współdzielona, lub adresem, który jest unikatowy dla procedury obsługi przerwania. Przykładowo, może to być adres dowolnej struktury związanej z sterownikiem zawierającym tę procedurę. Jest on wymagany do poprawnego wyrejestrowania tej procedury oraz do rozpoznania przez procedurę, że powinna obsłużyć przerwanie.

[1] Jego zawartość można wyświetlić na ekranie poleceniem `cat /proc/interrupt`.

Rejestracja i wyrejestrowanie procedury obsługi przerwania

Do wyrejestrowania procedury obsługi przerwania programista tworzący sterownik urządzenia może użyć funkcji `free_irq()`, o następującym prototypie:

```
void free_irq(unsinged int irq, void *dev)
```

Pierwszym argumentem tej funkcji jest numer linii IRQ, drugim może być wartość `NULL`, jeśli linia nie jest współdzielona, lub ten sam adres, który został przekazany jako piąty argument funkcji `request_irq()`, gdy rejestrowana była procedura obsługi współdzielonej linii przerwania.

Rejestracja i wyrejestrowanie procedury obsługi przerwania

Wątki przerwań (ang. *Threaded interrupts*)

W wersji 2.6.29 jądra wprowadzoną nowy sposób obsługi wątków. Są to tzw. wątki przerwań zapożyczone z gałęzi kodu jądra przeznaczonej dla rygorystycznych systemów czasu rzeczywistego. Głównym przesłaniem za zastosowaniem tego rozwiązania jest to, że procedura obsługi przerwania powinna się zakończyć najszybciej jak to możliwe, ponieważ nie może przejść w stan oczekiwania, a pozostałe prace związane z obsługą przerwania są wykonywane przez specjalny wątek jądra. Aby dane przerwanie było w ten sposób obsługiwane należy zarejestrować odpowiednią procedurę obsługi przerwania i wątek jądra za pomocą funkcji `request_threaded_irq()` o następującym prototypie:

```
int request_threaded_irq(unsigned int irq,
irq_handler_t handler, irq_handler_t thread_fn,
unsigned long flags, const char *name, void *dev)
```

Rejestracja i wyrejestrowanie procedury obsługi przerwania

Wątki przerwań (ang. *Threaded interrupts*)

Przyjmuje ona te same argumenty jak `request_irq()` oraz jeden dodatkowy (trzeci w kolejności), którym jest wskaźnik do funkcji wątku jądra. Procedura obsługi przerwania powinna być zarejestrowana z użyciem flagi `IRQF_ONESHOT`.

Procedury obsługi przerwań

W przypadku Linuksa procedury obsługi przerwań są funkcjami napisanymi w języku C, ale mogą także zawierać wstawki asemblerowe. Ich definicje są umieszczane w sterownikach urządzeń odpowiedzialnych za obsługę urządzeń peryferyjnych, które najczęściej są implementowane w postaci modułów jądra. Prototyp procedury obsługi przerwania jest następujący:

```
static irqreturn_t intr_handler(int irq, void *dev)
```

Nazwa rzeczywistej procedury obsługi przerwania powinna być inna niż podana w tym prototypie. Przez pierwszy parametr przekazywany jest numer przerwania. Wartość drugiego parametru jest istotna tylko jeśli procedura jest zarejestrowana dla współdzielonej linii IRQ. To ten sam unikatowy adres użyty podczas jej rejestracji. Typ `irqreturn_t` jest zdefiniowany przy użyciu słowa kluczowego `typedef` i w zależności od wersji jądra może być typem `int` lub `void`. To rozwiązanie wprowadzono, aby zachować kompatybilności istniejących sterowników z nowszymi i starszymi wersjami jądra.

Procedury obsługi przerwań

Procedura obsługi przerwania może zwracać następujące wartości: `IRQ_NONE` — przerwanie nie zostało przez nią obsłużone, `IRQ_HANDLED` — przerwanie zostało przez nią obsłużone. Aby uprościć zwracanie tych wartości programiści jądra stworzyli makro `IRQ_RETVAL(x)`, które jest rozwijane do wartości `IRQ_HANDLED` kiedy jego argument jest liczbą różną od zera i do wartości `IRQ_NONE`, kiedy ten argument to 0. Procedura obsługi przerwania związana z wątkiem przerwania może zwrócić wartość `IRQ_WAKE_THREAD`, która powoduje, że jądro aktywuje wątek obsługi danego przerwania. W przeszłości procedury obsługi przerwań miały dodatkowy parametr, który był wskaźnikiem do struktury typu `struct pt_regs`. Okazało się jednak, że większość takich procedur nie potrzebuje wartości rejestrów, więc go usunięto. Procedury, które potrzebują wartości rejestrów mogą uzyskać adres wspomnianej struktury za pomocą funkcji `get_irq_regs()`.

Procedury obsługi przerwań

Procedury obsługi przerwań w Linuksie wykonywane są w *kontekście przerwania*. Oznacza to, że ich działanie podlega kilku ograniczeniom. Przede wszystkim muszą one działać szybko, ponieważ obsługa przerwania może wstrzymywać inne istotne operacje w jądrze lub przestrzeni użytkownika. Procedura obsługi przerwania nie jest związana z żadnym procesem, w związku z tym nie może wywołać żadnych funkcji powodujących przejście procesu w stan oczekiwania. Przykładowo nie może ona wywołać `request_irq()` celem rejestracji innej procedury obsługi przerwania. Aby poradzić sobie z tymi ograniczeniami kod odpowiedzialny za obsługę przerwań w Linuksie jest, podobnie jak w innych współczesnych systemach operacyjnych, podzielony na dwie części. Pierwsza jest nazywana *górną połówką* (ang. *top half*) a druga *dolną połówką* (ang. *bottom half*), mimo, że nie muszą być tego samego rozmiaru. W górnej połówce wykonywane są najważniejsze operacje związane z obsługą przerwania, które nie mogą zostać odłożone na później. Górna połówka to po prostu procedura obsługi przerwania.

Procedury obsługi przerwań

Pozostałe operacje są wykonywane w dolnej połówce. W Linuksie dostępnych jest kilka jej implementacji, z których większość będzie omawiana na następnych wykładach, ale jednym z przykładów są wątki przerwań. Wartość makra `current` nie jest przydatna dla procedur obsługi przerwań, które, jak już wspomniano, nie są związane z żadnym procesem, więc nie potrzebują deskryptora bieżącego procesu. Jednakże używają one stosu jądra procesu, tak jak inne funkcje jądra. Jego wielkość jest ograniczona do dwóch stron, więc w przypadku procesorów x86 ma on rozmiar 8KiB, a dla procesorów Alpha, 16KiB. Istnieją także poprawki dla kodu jądra, stosowane w przypadku systemów MPP (ang. *Massively Parallel Processing*), które zmniejszają wielkość tego stosu do jednej strony. W tym jednak przypadku procedury obsługi przerwań otrzymują osobny stos.

Procedury obsługi przerwań

Te procedury nie muszą być wielobieżne (ang. *reentrant*), bo jądro Linuksa nie obsługuje zagnieżdżonych przerwani, tzn. przerwań, które mogą się pojawić, gdy ich wcześniejsze instancje są obsługiwane. Jednakże w systemach wieloprocesorowych procedury obsługi przerwań mogą wymagać zastosowania środków synchronizacji, jeśli używają współdzielonych zasobów.

Przerwania sygnalizowane wiadomością

Współczesne urządzenia, używające takich magistral jak USB, PCI, PCI–Express potrzebują dużej liczby przerwań, które są przydzielane im dynamicznie (niektóre przerwania są przydzielane statycznie z powodów historycznych). To oznacza, że wiele linii IRQ musiałoby być współdzielone przez te urządzenia, co prowadzi do wielu problemów. Aby je rozwiązać inżynierowie zajmujący się sprzętem wprowadzili tzw. *przerwania sygnalizowane wiadomością* (ang. *Message Signaled Interrupts* — MSI). Te przerwania nie są zgłaszane za pomocą zmiany stanu fizycznej linii IRQ, ale poprzez zapis krótkiej wiadomości (kilka bajtów) pod określony adres w pamięci. Pierwsze rozwiązanie tego typu wprowadzono w standardzie 2.2 magistrali PCI. W jego wersji 3.0 dodano możliwość maskowania pojedynczych takich przerwań, jak również umożliwiono urządzeniom na posiadanie kilku z nich, niezależnie skonfigurowanych. To rozwiązanie nazwano MSI–X. Jądro Linuksa posiada API do obsługi przerwań sygnalizowanych wiadomością począwszy od wersji 4.8.

Przerwania sygnalizowane wiadomością

W skład tego API wchodzą trzy funkcje:

`pci_alloc_irq_vectors()` przydziela wektory przerwań dla urządzeń PCI. Pobiera ona cztery argumenty. Pierwszym jest adres struktury typu `struct pci_dev`, związanej z tym urządzeniem, drugim jest minimalna liczba wektorów (jeśli jest wymagana), trzecim jest maksymalna liczba wektorów. Ostatnim jest jedna flaga lub ich suma bitowa. W przypadku powodzenia funkcja zwraca 0, a w przeciwnym przypadku wartość `-ENOSPC`.

`pci_irq_vector()` przypisuje urządzeniu PCI numer przerwania. Ta funkcja przyjmuje dwa argumenty, adres struktury typu `struct pci_dev` i numer przerwania. Zwraca ona 0 w przypadku powodzenia i wartość różną od 0 w przeciwnym razie.

`pci_free_irq_vectors()` zwalnia przydzielone wektory przerwań. Ta funkcja nie zwraca żadnej wartości, a jako argument przyjmuje adres struktury typu `struct pci_dev`.

Przerwania sygnalizowane wiadomością

Do `pci_alloc_irq_vectors()` mogą być przekazane następujące flagi:

`PCI_IRQ_LEGACY` urządzenie PCI będzie używało przerwań sygnalizowanych przy pomocy linii IRQ, a nie MSI (tryb domyślny),

`PCI_IRQ_MSI` urządzenie PCI będzie używało podstawowych MSI,

`PCI_IRQ_MSIX` urządzenie PCI będzie używało MSI–X,

`PCI_IRQ_ALL_TYPES` urządzenie PCI będzie używało dowolnego dostępnego typu przerwań,

`PCI_IRQ_AFFINITY` w systemach wieloprocesorowych funkcja rozdzieli przerwania między wszystkie dostępne CPU.

Funkcja `pci_irq_vector()` jest używana do uzyskania numeru przerwania, wymaganego do rejestracji procedury obsługi za pomocą `request_irq()` lub `request_threaded_irq()`.

Sterowanie systemem przerwań

Następujące funkcje i marka jądra są używane do sterowania systemem przerwań:

`local_irq_disable()` wyłącza lokalny system przerwań,

`local_irq_enable()` włącza lokalny system przerwań,

`local_irq_save(unsigned long flags)` zapisuje bieżący stan systemu przerwań i go wyłącza,

`local_irq_restore(unsigned long flags)` przywraca zapisany stan systemu przerwań,

`disable_irq_nosync(unsigned int irq)` blokuje daną linię IRQ i natychmiast kończy swoje wykonanie,

`disable_irq(unsigned int irq)` wyłącza daną linię IRQ, a jeśli jakaś procedura obsługi wykonuje się dla tej linii, to czeka na jej zakończenie,

`enable_irq(unsigned int irq)` włącza linię IRQ wyłączoną przez `disable_irq_nosync()`,

Sterowanie systemem przerwań

`synchronize_irq(unsigned int irq)` włącza linię IRQ wyłączoną przez `disable_irq()`,

`irqs_disabled()` zwraca wartość różną od 0, jeśli lokalny system przerwań jest wyłączony,

`in_interrupt()` zwraca 0 w kontekście procesu i wartość różną od 0 w kontekście przerwania,

`in_irq()` zwraca wartość różną od 0, jeśli zostanie wywołana w procedurze obsługi przerwania, a 0 w przeciwnym przypadku.

Funkcja `synchronize_irq()` musi być wywołana tyle razy, ile była wywołana `disable_irq()`. Podobna zależność występuje w przypadku `enable_irq()` i `disable_irq_nosyc()`.

Pytania

?

Koniec

Dziękuję Państwu za uwagę!