Wykład szósty

Obsługa przerwań

Jednym z zadań systemu operacyjnego jest komunikacja z urządzeniami wejścia – wyjścia. Ponieważ są one zazwyczaj dużo wolniejsze od procesora, to zwykle nie oczekuje on na zakończenie przez nie działania. Mimo to procesor powinien wiedzieć kiedy taki moment nastąpi. Jednym z rozwiązań tej kwestii jest okresowe sprawdzanie stanu takich urządzeń, czyli tzw. *polling*. Niestety, to rozwiązanie może prowadzić do sytuacji, w której system operacyjny może „przegapić" informację, którą przesłało urządzenie. Lepszym rozwiązaniem jest zastosowanie systemu przerwań. Przerwanie jest sygnałem dla procesora, że oto jakieś urządzenie zewnętrzne wykonało swoją pracę i należy je odpowiednio obsłużyć. Przerwania są najczęściej generowane asynchronicznie – mogą pojawić się w dowolnym momencie pracy procesora. Każde urządzenie dysponuje połączeniem z kontrolerem przerwań. Kiedy chce ono zgłosić przerwanie sygnalizuje to za pomocą tego właśnie połączenia. Kontroler powiadamia o wystąpieniu przerwania procesor. Każdemu urządzeniu jest również przyporządkowany numer przerwania, który pozwala określić, które urządzenie zgłosiło przerwanie i jak to przerwanie należy obsłużyć. Numer przerwania skojarzony jest z linią zgłoszenia przerwania IRQ (*ang. Interrupt Request*). W komputerach kompatybilnych z IBM PC część przerwań jest na stałe przypisana pewnym urządzeniom, a część – w szczególności dla tych urządzeń, które są podłączone z systemem przez magistralę PCI lub nowsze szyny, takie jak PCI-Express, USB – jest przydzielana w sposób dynamiczny. Oprócz obsługi urządzeń zewnętrznych system przerwań pozwala na obsługę sytuacji wyjątkowych.

Z każdym przerwaniem, które jest skojarzone z urządzeniem lub wyjątkiem związana jest procedura obsługi takiego przerwania (*ang. interrupt handler* lub *interrupt service routine – ISR*), nazywana *górną połówką*. W przypadku Linuksa procedury te są po prostu funkcjami napisanymi w języku C i umieszczonymi w sterowniku danego urządzenia lub w części jądra odpowiedzialnej za obsługę wyjątku. Każda z takich funkcji jest napisana zgodnie z określonym prototypem. Od zwykłych funkcji różni je jedynie to, że wykonywane są w kontekście przerwania i wyłącznie w reakcji na pojawienie się sytuacji krytycznej lub sygnału od urządzenia. Procedura obsługi przerwania wywoływana jest w sposób asynchroniczny i dlatego ważnym jest, aby jej wykonanie zostało zakończone w jak najkrótszym czasie. Z tego powodu kod obsługi przerwań jest podzielony na dwie części zwane *górną połówką* i *dolną połówką*. Górna połówka zajmuje się wszystkimi czynnościami, których wykonanie jest konieczne zaraz po odebraniu sygnału przerwania, przede wszystkim powiadamia urządzenie o przyjęciu przerwania. Połówka dolna realizuje wszystkie te czynności, których wykonanie można odroczyć na pewien czas. Zazwyczaj jednak połówki dolne są wykonywane zaraz po wykonaniu połówek górnych.

Każda procedura obsługi przerwania musi zostać zarejestrowana za pośrednictwem funkcji *request_irq*, która kojarzy funkcję z przerwaniem i uaktywnia daną linię:

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

Pierwszy parametr określa numer przerwania, drugi jest wskaźnikiem na procedurę obsługi przerwania[1], trzeci parametr albo jest zerem, albo maską bitową (sumą bitową) mogącą się składać ze znaczników będących flagami. Cztery najważniejsze z nich to: IRQF_DISABLED – określa czy dana procedura jest szybką procedurą obsługi przerwania (od wersji 4.1 wycofywana z użycia), jeśli tak, to jest ona wykonywana przy wyłączonych wszystkich przerwaniach w systemie, IRQF_TIMER – procedura obsługuje przerwanie zegarowe, IRQF_SAMPLE_RANDOM – określa, czy przerwanie zasili pulę entropii jądra (od wersji 3.6 jądra nie jest używana), IRQF_SHARED – określa możliwość współdzielenia linii przerwania z innymi procedurami obsługi przerwań. Każda z nich musi być rejestrowana z tym znacznikiem ustawionym. Czwarty parametr, to nazwa urządzenia, która jest używana w katalogu */proc/irq* i w pliku */proc/interrupts*. Ostatni parametr jest używany w obsłudze linii współdzielonych. Pozwala on między innymi na zwolnienie z linii tylko jednej procedury obsługi. Jeśli linia nie jest współdzielona, to przez ten parametr przekazywana jest wartość NULL, ale w przypadku linii współdzielonych przekazywany jest unikalny adres identyfikujący sterownik urządzenia, który może być wykorzystywany przez procedury obsługi przerwań. Funkcja *request_irq()* zwraca zero, jeśli jej wykonanie zakończy się sukcesem. Wyjątek wykonania jest sygnalizowany wartością -EBUSY. Funkcja ta może ulegać blokowaniu, więc nie może być użyta w kontekście przerwania.

Komplementarną do *request_irq()* jest funkcja *free_irq()*:

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

Funkcja ta zwalnia linię IRQ określoną wartością parametru „irq". W przypadku linii współdzielonych konieczne jest określenie unikalnego adresu związanego z sterownikiem urządzenia za pomocą drugiego parametru, gdyż z linii usuwana jest tylko procedura związana z tym urządzeniem. W przypadku linii niewspółdzielonych można przekazać tym parametrem NULL.

Funkcje obsługi przerwań są definiowane według następującego prototypu:

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

Pierwszy parametr określa numer przerwania. Przez drugi przekazywany jest unikalny adres identyfikujący sterownik urządzenia, które współdzieli z innymi urządzeniami linię przerwania. Tą wartością może być adres arbitralnie wybranej struktury sterownika, która może być przydatna podczas działania funkcji. Nie wymaga się, aby funkcje te były wielobieżne, gdyż wywołanie jednej z nich powoduje zablokowanie linii z nią związanej. Funkcje te mogą zwracać dwie wartości: IRQ_HANDLED i IRQ_NONE. W ich miejsce można wykorzystać makrodefinicję IRQ_RETVAL(x), która zwraca IRQ_HANDLED jeśli parametr *x* jest różny od zera i IRQ_NONE w przeciwnym przypadku. Typ wartości zwracanej przez funkcję został wprowadzony celem zachowania kompatybilności z poprzednimi wersjami jądra. Jeśli podczas obsługi przerwania potrzebny jest dostęp do zawartości rejestrów procesora sprzed zgłoszenia przerwania, to funkcja obsługi przerwania może go otrzymać przy pomocy funkcji *get_irq_regs()*, która zwraca wskaźnik na strukturę *struct pt_regs*.

Procedury obsługi przerwań są wywoływane w kontekście przerwania, co oznacza, że nie są dozwolone w nich wywołania funkcji blokujących oraz nie jest przydatna (poza procedurami obsługi wyjątków) wartość zwracana przez makrodefinicję *current*. Procedury obsługi przerwań muszą wykonywać się szybko, aby nie blokować kolejnych zgłoszeń przerwania na tej samej linii. Korzystają one ze stosu jądra, który jest ograniczony do 8KB w 32 – oraz 64 – bitowych platformach sprzętowych PC i do 16KB w 64 – bitowych platformach sprzętowych Alpha.

W chwili otrzymania zgłoszenia przerwania jądro zapamiętuje bieżące wartości rejestrów na stosie i wywołuje funkcję *do_IRQ*(). Ta funkcja zapamiętuje zawartość rejestrów w strukturze typu *struct pt_regs*, odczytuje z nich numer przerwania, a następnie potwierdza jego przyjęcie i blokuje linię z nim związaną wywołując *mask_and_ack_8259A*()[2]. Dalej *do_IRQ*() sprawdza w tablicy deskryptorów przerwań o nazwie *irq_desc*, czy linia posiada jakąś zarejestrowaną procedurę obsługi. Kolejną czynnością jest wywołanie *handle_irq_event()*. Ta funkcja przegląda listę wszystkich procedur związanych z daną linią i po kolei je uruchamia. Któraś z nich powinna zwrócić wartość IRQ_HANDLED, sygnalizując tym samym, że obsłużyła przerwanie. Po zakończeniu przeglądania listy procedur obsługi przerwań, w starszych wersjach jądra, jeśli był ustawiony znacznik IRQF_SAMPLE_RANDOM dla funkcji, która obsłużyła przerwanie, to *handle_irq_event*() wywoływała *add_interrupt_randomness(),* a w nowszych zawsze wywołuje tę funkcję. W obu przypadkach kończy ona swe działanie i sterowanie wracała do *do_IRQ()*, która porządkuje stos i wywołuje *ret_from_intr().* Ta ostatnia wykonuje czynności związane z przywróceniem pracy procesu użytkownika bądź kodu jądra.

1 Jego typ zdefiniowany jest następująco: typedef irqreturn_t (*irq_handler_t) (int, void *);

2 To odnosi się tylko do 32-bitowych komputerów klasy PC, inne klasy komputerów inaczej blokują przerwania, mogą mieć inne prototypy *do_IRQ*(), lub nie korzystają z niej wcale (mają inną funkcję wykonującą podobne czynności).

Informacje statystyczne na temat przerwań obsługiwanych przez poszczególne procesory w systemie można znaleźć w pliku */proc/interrupts*. Są tam informacje o liczbie obsłużonych przerwań na danej linii, o urządzeniu lub urządzeniach, które są skojarzone z tą linią, oraz o kontrolerze przerwań, który obsługuje zgłoszenie danego przerwania.

Do obsługi systemu przerwań w jądrze zdefiniowano następujące funkcje/makrodefinicje:

1. *local_irq_disable()* - wyłącza lokalny (dla jednego procesora) system przerwań,
2. *local_irq_enable()* - komplementarna względem poprzedniej funkcji,
3. *local_irq_save(unsigned long flags)* - zachowuje bieżący stan przerwań,
4. *local_irq_restore(unsigned long flags)* - przywraca zapamiętany stan przerwań,
5. *disable_irq_nosync(unsigned int irq)* – natychmiastowe wyłączenie przerwania o zadanym numerze,
6. *disable_irq(unsigned int irq)* – jak wyżej, ale z odczekaniem, aż zakończone zostaną wywołania procedur obsługi przerwań skojarzonych z tą linią,
7. *enable_irq(unsigned int irq)* – odblokowanie przerwania (komplementarna do *disable_irq_nosync()*),
8. *synchronize_irq(unsigned int irq)* – jak wyżej (komplementarna do *disable_irq()*),
9. *irqs_disabled()* - sprawdzenie stanu lokalnego systemu przerwań,
10. *in_interrupt()* - określenie kontekstu wykonywanego kodu,
11. *in_irq()* - określenie, czy aktualnie wykonywany kod jest realizowany w ramach procedury obsługi przerwania.

Funkcje *local_irq_disable()* i *local_irq_enable()* zastąpiły funkcje *cli()* i *sti()*, które wyłączały system przerwań dla wszystkich procesorów dostępnych w systemie i nie były w związku z tym optymalne. Działanie tych funkcji nie jest bezpieczne, więc należy zapamiętać i potem przywrócić stan przerwań za pomocą pary następnych funkcji. Funkcje *disable_irq_nosync()* i *enable_irq()* oraz *disable_irq()* i *synchronize_irq()* muszą być wykonywane naprzemiennie, tzn. jeśli np. funkcja *disable_irq_nosync()* została wykonana określoną liczbę razy, to tyle samo razy musi być wywołana funkcja *enable_irq()*.

W kolejnych wersjach jądra Linuksa serii 2.6 wprowadzono szereg zmian do górnych połówek obsługi przerwań. W wersjach jądra wcześniejszych niż 2.6.20 procedury obsługi przerwań przyjmowały dodatkowy parametr, jakim był wskaźnik, na strukturę *struct pt_regs*. Ponieważ był on używany przez niewiele funkcji obsługi przerwań, to zdecydowano o jego usunięciu, celem zaoszczędzenia miejsca na stosie. Dla tych, które potrzebują wartości rejestrów zdefiniowano funkcję o nazwie *get_irq_regs()*, która zwraca wskaźnik na strukturę *struct pt_regs*. Do czasu wydania wersji 2.6.22 jądra dostępne były tylko trzy flagi, które były wykorzystywane podczas rejestracji procedury obsługi przerwania: SA_INTERRUPT dla przerwań szybkich i przerwania zegarowego, SA_SAMPLE_RANDOM oraz SA_SHIRQ. W jądrze o numerze 2.6.29 wprowadzono mechanizm wątków przerwań. Celem tego mechanizmu jest umieszczenie obsługi przerwania w osobnym wątku jądra, eliminując tym samym niedogodności związane z brakiem blokowania w kontekście przerwania oraz niektóre mało wygodne w obsłudze dolne połówki. Ten mechanizm jest opcjonalny, aby uniknąć konieczności zmiany kodu wszystkich sterowników urządzeń. Procedury obsługi przerwania, które mają być wykonane w ramach wątku rejestrowane są za pomocą funkcji 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)

W porównaniu z funkcją rejestrującą zwykłe procedury obsługi przerwania przyjmuje ona dodatkowy parametr, będący wskaźnikiem na funkcję wątku. Procedura obsługi przerwania, której adres jest przekazywany przez parametr *handler*, może oprócz tych samych wartości, które były wcześniej opisane, zwrócić IRQ_WAKE_THREAD, która nakazuje aktywację wątku odpowiedzialnego za obsługę przerwania.

Aby rozwiązać problem konieczności współdzielenia linii zgłoszeń przerwań przez wiele urządzeń, inżynierowie projektujący magistralę PCI wprowadzili rozwiązanie nazwane MSI (ang. *Message Signaled Interrupts*). Dzięki niemu urządzenia wejścia-wyjścia mogą zgłaszać przerwania zapisując specjalną wartość do miejsca w pamięci operacyjnej, o określonym adresie. Nie potrzebują zatem w ogóle linii IRQ. To udogodnienie pojawiło się po raz pierwszy w wersji 2.2 standardu PCI. W kolejnej wersji (3.0) dodano możliwość indywidualnego maskowania tych przerwań, oraz indywidualnej konfiguracji kilku przerwań MSI dla pojedynczego urządzenia. To drugie udogodnienie nazwano skrótem MSI-X.
Począwszy od wersji 4.8 jądro Linuksa udostępnia następujące funkcje do obsługi przerwań MSI:

*int pci_alloc_irq_vectors(struct pci_dev *dev, unsigned int min_vecs,unsigned int max_vecs, unsigned int flags);*

Ta funkcja alokuje wektory przerwań dla urządzenia. Jako pierwszy argument przyjmuje adres struktury będącej deskryptorem urządzenia PCI, jako drugi minimalną liczbę wektorów do zaalokowania, a jako trzeci maksymalną liczbę tych wektorów. Czwarty argument to pojedyncza flaga lub suma bitowa flag, które nie wykluczają się wzajemnie. Do tych flag należą: PCI_IRQ_LEGACY – urządzenie będzie używało zwykłych przerwań, zamiast MSI (domyślny tryb działania funkcji), PCI_IRQ_MSI – urządzenie będzie używało podstawowych przerwań MSI, PC_IRQ_MSX – urządzenie będzie używało przerwań MSI-X, PCI_IRQ_ALL_TYPES – urządzenie będzie używało tego rodzaju przerwań, który jest dostępny, PCI_IRQ_AFFINITY – w systemach wieloprocesorowych przerwania będą rozsyłane do wszystkich dostępnych procesorów. Funkcja stara się przydzielić maksymalną liczbę wektorów przerwań, która jest określona jako argument jej wywołania. W przypadku gdy to jej się powiedzie zwróci zero, a w przeciwnym przypadku wartość -ENOSPC.

*int pci_irq_vector(struct pci_dev *dev, unsigned int nr);*

Ta funkcja pozwala każdemu wektorowi przerwania MSI związanemu z urządzeniem PCI przypisać numer przerwania, aby można było dla niego zarejestrować procedurę obsługi przerwania przy pomocy *request_irq*(). Jako pierwszy argument wywołania ta funkcja przyjmuje adres deskryptora urządzenia PCI (struktury go opisującej), a jako drugi numer wektora. Zwraca ona numer przerwania.

*void pci_free_irq_vectors(struct pci_dev *dev);*

Ta funkcja zwalnia wektory przerwań MSI dla danego urządzania PCI. Jako argumenty wywołania przyjmuje adres deskryptora tego urządzenia.

Obsługa przerwań MSI została po raz pierwszy dodana w wersji 2.6.29 jądra. Z punktu widzenia programistów sterowników urządzeń najważniejszym jej elementem była funkcja:

*int pci_enable_msi_block(struct pci_dev *dev, int count)*

która pozwalała sterownikowi włączyć zgłaszanie bloków przerwań MSI. Później została ona zastąpiona przez następujący szereg funkcji: *pci_enable_msi*(), *pci_disable_msi*(), *pci_enable_msix_range*(), *pci_enable_msix_exact*() i *pci_disable_msix*(), które ostatecznie zostały zastąpione przez wcześniej opisane API dla wersji 4.8.