

## Wykład szósty

## Obsługa przerwania

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 przerwania. 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 przerwania. 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 przerwania 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 przerwania 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 przerwania 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<sup>1</sup>, trzeci parametr albo jest zerem, albo maską bitową (suma bitowa) mogąca 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 obsługująca przerwanie zegarowe, *IRQF\_SAMPLE\_RANDOM* – określa, czy przerwanie zasilii pulę entropii jądra (od wersji 3.6 jądra nie jest używana), *IRQF\_SHARED* – określa możliwość współdzielenia linii przerwania z innymi procedurami obsługi przerwania. Każda z nich musi być rejestrowana z tym znacznikiem ustawionym. Czwartym parametrem, 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 przerwania. 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 przerwania 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 przerwania 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 przerwania 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()*<sup>2</sup>. Dalej *do\_IRQ()* sprawdza w tablicy deskryptorów przerwania 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 przerwania, 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 procesora użytkownika bądź kodu jądra.

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

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

1. *local\_irq\_disable()* - wyłącza lokalny (dla jednego procesora) system przerwania,
2. *local\_irq\_enable()* - komplementarna względem poprzedniej funkcji,

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).

3. *local\_irq\_save(unsigned long flags)* - zachowuje bieżący stan przerwania,
4. *local\_irq\_restore(unsigned long flags)* - przywraca zapamiętany stan przerwania,
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 przerwania 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 przerwania,
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 wylączyły system przerwania 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 przerwania za pomocą pary następných 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 przerwania. W wersjach jądra wcześniejszych niż 2.6.20 procedury obsługi przerwania przyjmowały dodatkowy parametr, jakim był wskaźnik, na strukturę *struct pt\_regs*. Ponieważ był on używany przez niewiele funkcji obsługi przerwania, 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 przerwania szybkich i przerwania zegarowego, SA\_SAMPLE\_RANDOM oraz SA\_SHIRQ. W jądrze o numerze 2.6.29 wprowadzono mechanizm wątków przerwania. 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 zwykle 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. W tej wersji jądra wprowadzono także funkcję o prototypie:

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

która pozwala sterownikowi włączyć zgłaszanie bloków przerwania MSI (ang. *Message Signaled Interrupts*). Tego typu przerwania są wykorzystywane przez sprzęt korzystający z magistrali PCI-Express lub PCI 2.2. W przeciwieństwie do tradycyjnych przerwania, tego typu przerwania nie są zgłaszane poprzez zmianę stanu na odpowiedniej linii przerwania, lecz za pomocą zapisu przez urządzenie informacji o niewielkim rozmiarze pod określony adres w pamięci. Ta informacja jest odczytywana przez PIC (kontroler przerwania), który sygnalizuje przerwanie procesorowi. W nowszych wersjach jądra ta funkcja została zastąpiona przez dwie: *pci\_enable\_msi\_exact()* oraz *pci\_enable\_msi\_range()*.