Przerwania sprzętowe

20 grudnia 2000

1 Wprowadzenie.

Obsługa przerwań sprzętowych, w oczywisty sposób, mocno uzależniona jest od platformy na jakiej działa system. My zajmiemy się komputerami zgodnymi ze standardem PC AT, czyli zbudowanymi na bazie procesorów z rodziny i386 i wyposażonymi w kontroler przerwań zgodny z układem 8259A. Przerwania są generowane, gdy jedno z urządzeń wchodzących w skład systemu komputerowego, dojdzie do wniosku, że zaistniała sytuacja wyjątkowa, wymagająca wykonania pewnych czynności przez system operacyjny. System, żeby obsużyć takie zdarzenie, musie przerwać wykonywanie aktualnie liczonego kodu i zają się procedurą obsługi danego przerwania. Aby możliwy był następnie powrót do wykonywanych wcześniej zadań, przed rozpoczęciem obsługi zgłoszonego przerwania należy zapamiętać stan procesora. Kod procedury obsługi przerwania wykonywany jest w trybie nadzorcy, dlatego też w momencie wystąpienia przerwania w większości wypadków należy również przeączyć procesor do trybu uprzywilejowanego. Po zakończeniu procedury obsułgi przerwania należy wykonać czynności odwrotne, czyli przywrócić stan systemu z przed pojawienie się przerwania. Ze względu na sposób powstawania przerwania dzielimy na:

- Przerwania sprzętowe *(hardware interrupts)* zgłaszane są przez sterownik przerwań na żądanie jakiegoś z podączonych do niego urządzeń np. klawiatura, dysk twardy, modem.
- Wyjątki *(exceptions)* generowane są przez procesor w momencie gdy zaistnieje sytuacja wymagająca specjalnych dziaań. Wyjątki dzielimy na:
  - Niepowodzenia *(faults)* zgłaszane są przed zakończeniem wykonywania instrukcji, umożliwiają jej powtórzenie. Dobrym przykładem jest tu błąd braku strony *(Page Faults).*
  - Potrzaski *(traps)* generowane są po wykonaniu niektórych instrukcji procesora np. INT. W linuxie w ten właśnie sposób zrealizowano wywołania funkcji systemowych.
  - Załamania *(abords)* zgłaszane są gdy podczas obsługi niepowodzenia pojawi się następne.

W niniejszym opracowaniu zajmiemę się jedynie przerwaniami sprzętowymi gdyż wydaje mi się że kwestie wyjątków zostaną dokładnie wyjaśnione przy okazji omawiania systemowych mechanizmów związanych z ich obsługą.

2 Przerwania sprzętowe.

Przeledźmy teraz co dzieje się z kontrolerem przerwań od momentu pojawienia się sygnału żądania przerwania na jednym z jego wyprowadzeń do chwili odebrania sygnału EOI *(End Of Interrupt)*.

- Stan jednego z urządzeń podłączonych do kontrolera zmienia się na tyle, że jego elektronika dochodzi do wniosku, iż należy powiadomić o tym fakcie system operacyjny. Taką ważną sytuacją może by np. naciśnięcie klawisza na klawiaturze lub zakończenie transmisji dyskowej. Zgłaszane jest wtedy żądanie przerwania o numerze przypisanym do danego urządzenia.
- Kontroler przerwań sprawdza czy nie jest przypadkiem obsługiwane przerwanie o wyższym lub równym priorytecie (najwyższy priorytet ma zegar). Jeżeli jest to trzeba czekać aż wszystkie takie przerwania zostaną obsłużone. Obsługa przerwań o niższym priorytecie jest przerywana.
- Przerwania nie są zliczane. Oznacza to, że jednocześnie może czekać tylko jedno przerwanie o danym numerze, jeżeli pojawi się następne zostanie zignorowane. Jest to jeden z powodów, dla których procedury obsługi przerwań powinny wykonywać się możliwie krótko.
- Gdy z punktu widzenia kontrolera można już zgłosi przerwanie musi on poczekać, aż procesor dojdzie do takiego samego wniosku, co zostanie zasygnalizowane wystawieniem flagi IF.
- Sterownik przerwań będzie uważał przerwanie za obsługiwane do czasu nadejścia z procesora sygnału EOI.

3 Struktury danych jądra systemu związane z przerwaniami.

W linuxie mechanizm obsługi przerwań został zaprojektowany w sposób pozwalający na pracę na różnych platformach sprzętowych czyli z wykorzystaniem różnych kontrolerów przerwań. Programista musi dostarczyć jedynie funkcje obsługi danego sterownika. Adresy tych funkcji będą pamiętane w polach struktury *hw_interrupt_type*. Aby móc z jednym numerem przerwania skojarzyć więcej urządzeń, a co za tym idzie zarejestrować więcej funkcji obsługi mamy strukturę *irq_action*. Ostatnią z podstawowych struktur danych związanych z obsługą przerwań jest tablica *irq_desc[]*. Zawiera on elementy typu *irq_desc_t* które opisuja każde z przerwań. Rozmiar tej tablicy jest równy ilości przerwań. Typ *irq_desc_t* jest zdefiniowany w następujący sposób:

```
typedef struct {
unsigned int status;
struct hw_interrupt_type *handler;
struct irqaction *action;
unsigned int depth; unsigned int unused[4];
} irq_dest_t
```

Znaczenie poszczególnych pól:

- *status* - status linii IRQ.
- *handler*- wskaźnik do struktury *hw_interrupt_type* opisującej kontroler przerwań związany z przerwaniem o danym numerze.
- *action* - wskaźnik do listy struktur *irq_action* opisujących zarejestrowane funkcje obsługi przerwań skojarzone z przerwaniem o danym numerze.

Definicja struktury *hw_interrupt_type* wygląda tak:

```
struct hw_interrupt_type {
const char *typename; void (*startup)(unsigned int irq);
void (*shutdown)(unsigned int irq);
void (*handle)(unsigned int irq, struct pt_regs regs);
void (*enable)(unsigned int irq);
void (*disable)(unsigned int irq);
}
```

Pola tej struktury to wskaźniki do funkcji odpowiedzialnych za obsługę danego kontrolera przerwań. Najważniejszą z nich jest funkcja zapamiętana w polu *handle*. Jest ona wywoływana w momencie wystąpienia przerwania. Dla komputerów PC z kontrolerem zgodnym z 8259A jest nią funkcja *do_8259A_IRQ()*, zostanie ona opisana w następnej części tego opracowania. Wydaje mi się, że nazwy pozostałych pól same tłumaczą zastosowanie funkcji do których wskaźniki są w nich pamiętane. Może warto jeszcze zaznaczyć, że dla kontrolera 8259A w polach *startup* i *enable* oraz *shutdown* i *disable* pamiętane są wskaźniki do tych samych funkcji.

Definicja struktury *irq_action* jest następująca:

```
struct irqaction {
void (*handler)(int irq, void *dev_id, struct pt_regs *regs);
unsigned long flags;
unsigned long mask;
const char *name;
void *dev_id;
struct irq_action *next;
}
```

W polu *handler* pamiętany jest adres funkcji obsługi. W pozostłay polach zapisane są parametry z jakimi będzie wywołana ta funkcja. Zostały one zapamiętane w momencie rejestrowania funkcji obsługi na podstawie parametrów przekazanych do funkcji rejestrującej *request_irq()* (o rejestrowaniu funkcji obsługi napiszę później). Ponieważ z jednym przerwaniem może być powiązanych kilka urządzeń a więc również kilka funkcji obsługi przerwania, w polu *next* mamy wskaźnik do kolejnej struktury *irqaction* opisującej następną funkcje obsługi.

4 Przepyw sterowania w momencie pojawienia się przerwania.

Opiszę teraz działanie funkcji które są wykonywane w momencie pojawienia się przerwania. W części dotyczącej funkcji obsługi kontrolera przerwań opis ten będzie dotyczły funkcji *do_8259A_IRQ()*. Reszta z opisywanych funkcji jest niezależna od platformy sprzętowej.

1. Wektory przerwań wskazują na funkcje tworzone makrem *BUILD_IRQ()* które:
   (a) zapamiętują rejestry procesora.
   (b) odkładają na stos numer przerwania.
   (c) wołają *do_irq()*.
   (d) wykonują algorytm *ret_from_intr*.
2. Funkcja *do_IRQ (struct pt_regs*)*
   (a) W "subtelny" sposób pobiera numer przerwania ze stosu.
   (b) Zwiększa licznik obsużonych przerwań o danym numerze na danym procesorze.
   (c) Woła funkcje *irq_desc[numer_przerwania].handler->handle()* przekazując jej numer przerwania i wskaźnik na strukturę z zapamiętanymi rejestrami. Dla PC AT funkcją tą jest *do_8259_IRQ()*.
   (d) Wykonuje zaznaczone do wykonania dolne połowy *(bottom halfs)* wołając funkcje *do_bottom_half()*.
3. Funkcja *do_8259A_IRQ (int, struct pt_regs*)*
   (a) Jeżeli dla danego przerwania jest ustawiona flaga *IRQ_DISABLED* lub *IRQ_INPROGRES* to kończy swoje działanie.
   (b) Ustawia flagę *IRQ_INPROGRES*.
   (c) Woła funkcje *handle_IRQ_event()*.
   (d) Zeruje flagę *IRQ_INPROGRES*.

(e) Jeżeli przerwanie nie ma ustawionej flagi *IRQ_DISABLED* to odblokowuje je.

4. Funkcja *handle_IRQ_event(int, struct pt_regs*, irqaction*)*

   (a) Dla każdej z zarejestrowanych funkcji obsługi:

      i. Odpowiednio ustawia flagę IF procesora w zależności od flagi *SA_INTERRUPT* funkcji obsługi.

      ii. Włoa funkcje obsługi.

   (b) Jeżeli która z funkcji obsługi miała ustawioną flagę *SA_SAMPLE_RANDOMI* to włoa funkcje *add_interrupt_randomes()*. Rozwiązanie to pozwala zwiększyć losowość systemowego generatora liczb pseudolosowych.

   (c) Zeruje flagę IF procesora.

5. Algorytm *ret_from_intr*

   (a) Przywraca stan rejestrów procesora z przed wystąpienia przerwania i kończy obsługę przerwania.

5 Rejestrowanie funkcji obsługi.

Aby funkcje *irq_desc[].handler->handle()* miłay co robić należy zarejestrować funkcje obsługi przerwań. Dzieje się to zazwyczaj podczas startu systemu a także podczas ładowania modułów obsługi urządzeń. Do rejestrowania i odrejestrowywania funkcji obsługi sużą funkcje *request_irq()* i *free_irq()*.

Funkcja *request_irq()* jest odpowiedzialna za rejestrowanie funkcji obsługi. Przekazujemy do niej następujące argumenty:

- *unsigned int irq* - numer przerwania.
- *void (*handler)(int, void*, struct pt_regs*)* - adres funkcji obsługi.
- *unsigned long flags* - flagi.
- *const char *devname* - nazwa urządzenia.
- *void *dev_id* - identufikator.

Wydaje mi się, że komentarza mogą wymagać jedynie pola *flags* i *dev_id*. Pierwsze z nich to iloczyn flag *SA_** . Wypisywanie wszystkich tych flag jest chyba zbędne ale warto wiedzieć, że flaga *SA_INTERRUPT* pozwala zablokować przerwania na czas działania funkcji, *SA_SHIRQ* jest odpowiedzialna za wspódzielenie przerwań a ustawienie flagi *SA_SAMPLE_RANDOM* prawdopodobnie poprawi działanie generatora liczb pseudolosowych. Pole *dev_id* to wskaźnik na strukturę (albo zmienną) dowolnego typu. Zawartość tego pola zostanie przekazana do funkcji obsługi przerwania. Programista piszący tę funkcje może użyć tego pola aby zapewnić sobie dostęp do struktur danych związanych z konkretnym urządzeniem. Warto jeszcze dodać, że wywołanie funkcji rejestrującej wcale nie musi zakończyć się sukcesem. Może się tak zdarzyć np. wtedy gdy żądany numer przerwania jest już zajęty a nie wszystkie funkcje do niego przypisane zgadzają się na wspódzielenie przerwania lub gdy nie powiedzie się alokacja pamięci na strukturę *irqaction*.

Funkcja *free_irq(unsigned int irq, void *dev_id)* usuwa funkcje obsługi przerwania i w razie potrzeby (gdy ta bya ostatnią skojarzona z danym przerwaniem) blokuje przerwanie. Funkcja ta zawsze kończy się sukcesem.

6 Dolne poowy.

Funkcje obsługujące przerwania powinny zawsze wykonywać się stosunkowo szybko ponieważ w tym czasie mamy zablokowane w najlepszym przypadku przerwania o niższym priorytecie a w najgorszym wszystkie (wyzerowana flag IF procesora). Stan taki może prowadzić do utraty ważnych informacji bo przecież przerwania nie są zliczane oraz zaburzeń pracy systemu. Niestety, czasami zdarza się, że procedura obsługi przerwania musi wykonać jakieś czasochonne zadanie. Może to być np. przekopiowanie danych do bufora w pamięci. Zazwyczaj jednak czynnoci te nie muszą by wykonane natychmiast a na dodatek nie jest wymagane aby podczas ich realizacji były zablokowane przerwania. W celu obsługi takich sytuacji powstała koncepcja dolnych i górnych poówek funkcji obsługi przerwania. Górna połowa takiej podzielonej funkcji wykonywana jest natychmiast po pojawieniu się przerwania z możliwocią zablokowania na czas jej działania przerwań a dolna zaznaczana jest do wykonania w późniejszym czasie już z odblokowanymi przerwaniami. Wskaźniki do funkcji dolnych połów znajdują się w systemowej tablicy

*void (*bh_base[32])(void).*

Do rejestrowania dolnych połów służy funkcja

*init_bh(int,void(*rantime)(void)).*

Jej argumentami są numer pod jakim chcemy zarejestrować funkcje i wskaźnik na funkcje będącą dolną połówką przerwania. Za pomocą funkcji

mark_bh(int nr)

zaznaczamy zaś do wykonania funkcje zarejestrowaną po podanym numerem. Funkcje dolnych połów są wywoływane na końcu funkcji *do_IRQ()*.

Ponieważ celem wprowadzenia tego mechanizmu było umożliwienie wykonywania częci czynności związanych z obsługą przerwania z odblokowanymi przerwaniami dolne połowy mają własność atomicznoci. Oznacza to, że dopóki nie zostanie zakończona jedna z tych funkcji system nie rozpocznie wykonywania żadnej innej. Pozwala to na odblokowanie przerwań.

Kłopoty może sprawiać ograniczenie do 32 liczby dolnych połów jakie możemy zarejestrować. Rozwiązaniem tego problemu może być stosowanie mechanizmu kolejek zadań zamiast systemu dolnych połów.