Komputerowe systemy sterowania

Cw. 4 - obsluga przerwan w systemie wielozadaniowym czasu rzeczywistego

(oprac. Adam Piórkowski)

Obsluga przerwan sprzetowych w systemie QNX

Zgloszenie przerwania sprzetowego w systemie QNX jest przejmowane bezposrednio przez kernel. Przerywa on dzialanie aktualnie wykonujacego sie procesu / watku i wywoluje procedure obslugi przerwania ISR (Interrupt Service Routine). Procedura ta ma na celu dokonanie pewnych prostych operacji, np. odczyt danych ze sprzetu czy sprawdzenie pochodzenia przerwania. Jej priorytet jest wyzszy od priorytetów jakichkolwiek watków oprogramowania. W przypadku potrzeby przekazania informacji do programu wysyla sygnal SIGEV_INTR. Operacja ta zabiera nieznaczny czas procesora. Proces zainteresowany przerwaniem uruchamia watek blokujacy sie oczekiwaniem na sygnal. Watkowi temu mozna nadac dowolny priorytet, co umozliwia uprzywilejowana w stosunku do pozostalych procesów obsluge przerwania czy tez nieprzerwana prace wazniejszych procesów. Takie rozwiazanie jest zgodne z postulatami wymagan czasu rzeczywistego w stosunku do systemu operacyjnego. Istotna jest tez mozliwosc obslugi jednego przerwania sprzetowego przez wiele procesów. Podobnie jednym procesem mozna obslugiwac kilka przerwan.

Programista ma do wyboru dwa sposoby implementacji:

- uzycie zdarzen systemowych,
- stworzenie wlasnej procedury ISR.

Przypadek pierwszy jest prosty w implementacji. Wykorzystywana jest standardowa procedura ISR. Aplikacja zainteresowana obsluga przerwania powoluje watek do jego obslugi. Nadaje on sobie odpowiednie prawa dostepu do sprzetu, podpina pod konkretna linie przerwan wyslanie sygnalu do siebie i przechodzi w stan oczekiwania. W stanie tym nie zajmuje zasobu procesora. W przypadku nadejscia przerwania watek odbiera sygnal wykonujac zadane operacje az do ponownego przejscia w stan oczekiwania. Szkic kodu ilustrujacego implementacje problemu zamieszczony jest ponizej:

```
void * obsluga_przerwania (void * data)
{
        int id;
        ThreadCtl( _NTO_TCTL_IO, NULL);              // nadanie odpowiednich praw watkowi
        id = InterruptAttachEvent( nr_ przerwania, &event, 0)
        if(id == -1)
                return;                              // przerwania nie udalo sie przydzielic

         while (1)
        {
                InterruptWait (NULL, NULL);          // oczekiwanie na przerwanie

                //  w tym miejscu nalezy wykonac operacje zwiazane z obsluga przerwania

                InterruptUnmask(nr_ przerwania, id); // odblokowanie zródla przerwania
        }
}
```

Tab. 1

Przypadek drugi dotyczy sytuacji wymagajacych wlasnej procedury ISR. Procedura ta dziala bowiem na poziomie kernela; jest przez niego uruchamiana w trybie natychmiastowym. Dotyczy to sytuacji, w których z przerwaniem zwiazane sa przychodzace dane, które nalezy odebrac jak najszybciej. Wówczas watek obslugi nie musi miec wysokiego priorytetu. Procedura ISR moze byc takze wykorzystywana do filtrowania zródel przerwania w przypadku wiekszej ich liczby. Jej szkic implementacyjny podany jest ponizej:

```
const struct sigevent* procedura_isr (void *arg, int id)
{
        // w tym miejscu mozna umiescic sprawdzenie zródla przerwania,
        // wykonac odwolania do sprzetu czy zablokowac ponowna obsluge przerwania

        return (&event);
}
```

Tab. 2

Procedura obslugi przerwania w tej metodzie bedzie nieznacznie zmodyfikowana:

```
void * obsluga_przerwania (void * data)
{
        int id;
        ThreadCtl( _NTO_TCTL_IO, NULL);
        id = InterruptAttach( nr_przerwania, procedura_isr, NULL, 0, 0);
        if(id == -1)
        . . .
}
```

Tab. 3