PROGRAMOWANIE SYSTEMÓW CZASU RZECZYWISTEGO

LABORATORIUM

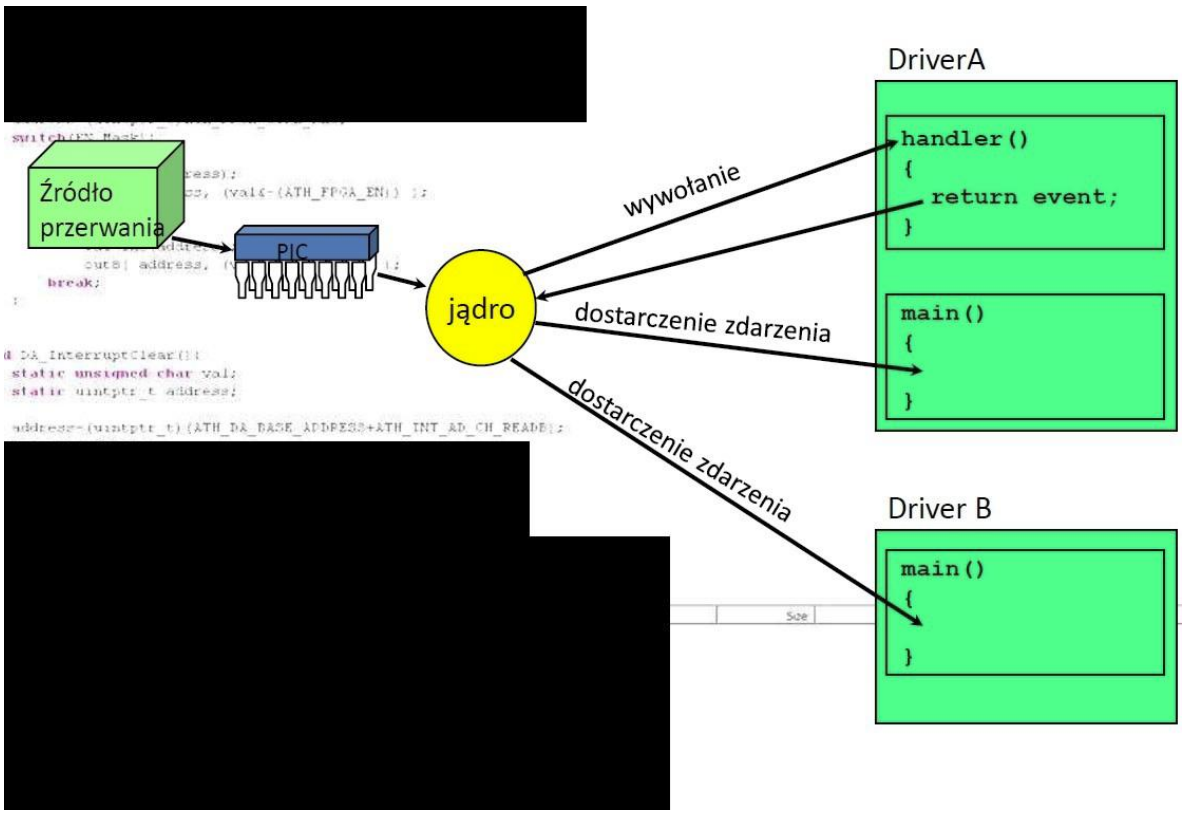
Temat: QNX Neutrino – Interrupts

Mariusz Rudnicki

2016

Wstęp

W QNX Neutrino wszystkie przerwania sprzętowe przechwytywane są przez jądro systemu.



Obsługę przerwania można zorganizować w dwojaki sposób:

- proces rejestruje funkcję obsługi przerwania ang. ***Interrupt Service Routine***, która zostanie wywołana po wystąpieniu przerwania;
- proces żąda od jądra systemu powiadomienia o wystąpieniu przerwania.

Wybór sposobu obsługi przerwania pociąga za sobą pewne konsekwencje z punktu widzenia systemu operacyjnego jak również innych procesów i wątków.

Właściwości funkcji obsługi przerwania:

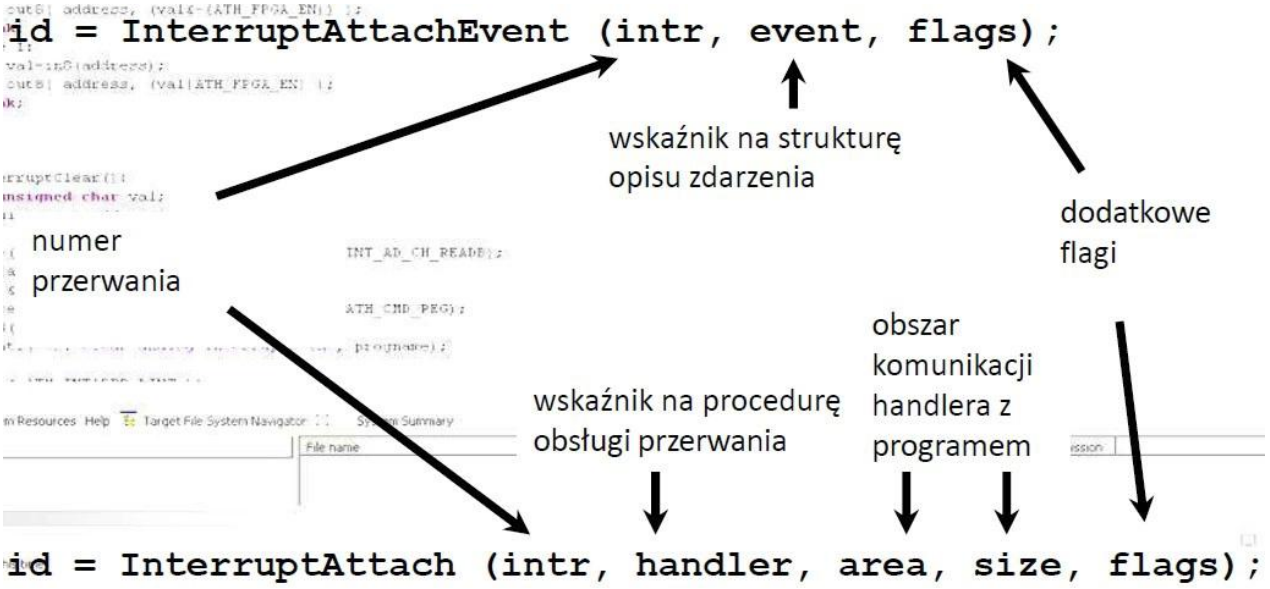
- rozmiar stosu, którym dysponuje procedura obsługi przerwania, jest ograniczony. Stąd nie powinna ona zawierać dużych tablic czy innych struktur danych. Bezpiecznie jest przyjąć, że dostępny rozmiar stosu wynosi około 200 bajtów.
- funkcja obsługi przerwań jest wykonywana asynchronicznie z wątkami należącymi do pewnego procesu i używa wspólnych z nimi danych. Wszystkie zmienne modyfikowane przez handler powinny być poprzedzone słowem kluczowym volatile a ich modyfikacja wewnątrz wątków zabezpieczona przez zablokowanie przerwań. Innym rozwiązaniem jest zastosowanie funkcji realizujących elementarne operacje arytmetyczne i bitowe w niepodzielny sposób. Nazwy tych funkcji zaczynają się od atomic_*.

- funkcja obsługi przerwania jest wykonywana poza normalnym szeregowaniem, więc powinna być krótka, jak to tylko możliwe. Jeżeli wymagane jest wykonanie czasochłonnych czynności, to powinny być one wykonane w wątku, który zostanie odblokowany przez handlera.
- funkcja obsługi przerwania nie może wywoływać żadnych funkcji systemowych poza sporadycznymi wyjątkami.

Porównanie ISR z powiadomieniem zdarzeniem

- rozmiar stosu, którym dysponuje procedura obsługi przerwania, jest ograniczony. Stąd nie powinna ona zawierać dużych tablic czy innych struktur danych.
- procedura obsługi przerwań jest wykonywana poza normalnym szeregowaniem, więc powinna być tak krótka, jak to tylko możliwe. Jeżeli wymagane jest wykonanie czasochłonnych czynności, to powinny być one wykonane w wątku, który zostanie przez handler odblokowany.
- odblokowany przez zdarzenie wątek ma priorytet i podlega zwykłemu szeregowaniu.
- uruchamianie wątków jest łatwiejsze niż handlerów.
- większa zwłoka pomiędzy przerwaniem a odblokowaniem wątku.
- w kodzie procedury obsługi przerwania należy wykonać tylko niezbędne czynności, a następnie powiadomić pewien wątek o wystąpieniu przerwania. Wątek ten wykona resztę pracy.

Główne funkcje wiążące przerwanie ze zdarzeniem lub funkcję obsługi przerwania z danym przerwaniem sprzętowym.



Przygotowanie platformy i środowiska IDE

Uruchom QNX Neutrino za pomocą maszyny wirtualnej. Sprawdź poleceniem ***pidin*** czy jest uruchomiony proces ***qconn***. Jeśli nie ma go na liście pracujących procesów uruchom go poleceniem ***qconn #***.Sprawdź adres IP poleceniem ***ifconfig***. Następnie uruchom środowisko QNX Momentics IDE. Wybierz przestrzeń roboczą wskazaną przez prowadzącego.

Zadanie 1.

Przeanalizuj kod źródłowy poniższego programu **int_timer**. Odszukaj w nim fragmenty odpowiedzialne za obsługę przerwań. Wypisz odpowiednie funkcje w sprawozdaniu.

```
/*
      int_timer.c
      This module demonstrates will contain code for handling an interrupt.
      To test is, simply run it. Note that you may have to do something to cause the
interrupts to be generated (e.g. press a key if handling the keyboard interrupt).
*/

#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/neutrino.h>
#include <sys/syspage.h>

char            *progname = "int_timer";
struct sigevent int_event;  // the event to wake up the thread

const struct sigevent *hdlr( void *blah, int id ){

  static int count = 0;
  if (++count > 1500 ){
    count = 0;
    return &int_event;
  }
  return NULL;
}

main (int argc, char **argv)
{
  int id;
  setvbuf (stdout, NULL, _IOLBF, 0);

  printf ("%s:  starting...\n", progname);

  // request I/O privity
  ThreadCtl(_NTO_TCTL_IO, 0 );
  // set up an event for the handler or the kernel to use to wake up
  // this thread.  Use whatever type of event and event handling you want
```

```
  SIGEV_INTR_INIT( &int_event );
  // either register an interrupt handler or the event

  id = InterruptAttach(0, hdlr, NULL, 0, _NTO_INTR_FLAGS_TRK_MSK );

  while (1) {
    // block here waiting for the event
    InterruptWait(0, NULL );
    // if using a high frequency interrupt, don't print every interrupt

    printf ("%s:  we got an event after 1500 timer ticks and unblocked\n", progname);
  }
}
```

Na podstawie powyższego kodu zmodyfikuj kod źródłowy projektu **int_timer** tak, aby mógł obsługiwać przerwania od timera systemowego. Po modyfikacjach uruchom program **int_timer**. Sprawdź w jakim stanie jest wątek oczekujący na przerwanie.

Zadanie 2.

Przeanalizuj kod źródłowy poniższego programu **int_kbd_event**. Odszukaj w nim fragmenty odpowiedzialne za obsługę przerwań. Wypisz odpowiednie funkcje w sprawozdaniu.

```
/*
int_kbd_event.c
      This module demonstrates will contain code for handling an interrupt.
      To test is, simply run it.  Note that you may have to do something to cause the
interrupts to be generated (e.g. press a key if handling the keyboard interrupt).
*/

#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/neutrino.h>
#include <sys/syspage.h>

char    *progname = "int_kbd_event";
struct sigevent int_event;  // the event to wake up the thread

main (int argc, char **argv)
{
  int id;
  int count = 0;

  setvbuf (stdout, NULL, _IOLBF, 0);

  printf ("%s:  starting...\n", progname);

  // request I/O privity
  ThreadCtl(_NTO_TCTL_IO, 0 );

  // set up an event for the handler or the kernel to use to wake up
  // this thread.  Use whatever type of event and event handling you want

  SIGEV_INTR_INIT( &int_event );
  // either register an interrupt handler or the event

  id = InterruptAttachEvent(1, &int_event, _NTO_INTR_FLAGS_TRK_MSK );

  while (1) {
    // block here waiting for the event
    InterruptWait(0, NULL );
    // if using a high frequency interrupt, don't print every interrupt
    InterruptUnmask( 1, id );
    printf ("%s:  we got a keyboard interrupt and unblocked (%d)\n",
            progname, count++);
  }
}
```

Na podstawie powyższego kodu zmodyfikuj kod źródłowy projektu **int_kbd_event** tak, aby mógł obsługiwać przerwania od klawiatury. Po modyfikacjach uruchom

program **int_kbd_event**. Sprawdź w jakim stanie jest wątek oczekujący na przerwanie. Podaj ile przerwań rejestrowanych jest podczas używania klawiatury. Skomentuj uzyskane wyniki.

Sprawozdanie.

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

… … … … … … … … … … … … … … … … … … … …

Grupa: