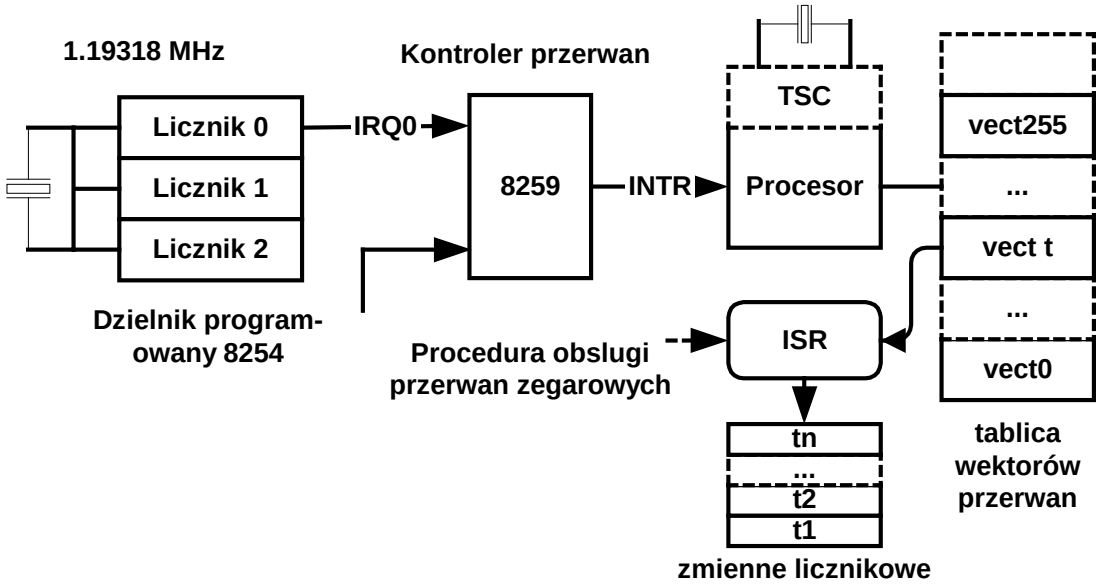
1 Timery i zdarzenia

Dokładność pomiaru czasu

Dokładność pomiaru czasu wynika ze sposobu zaprogramowania układów licznikowych 8254 które generują przerwania zegarowe



Rys. 1-1 Układy pomiaru czasu w komputerze PC

Dokładność pomiaru czasu tą metodą może być sprawdzona za pomocą funkcji **`clock_getres`**.

```
struct timespec {
  long tv_sec;   // sekundy
  long tv_nsec;  // nanosekundy
}

int clock_getres(clockid_t clk_id, struct timespec
                 *res);
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main( void )   {
    struct timespec res;

    if ( clock_getres( CLOCK_REALTIME, &res) == -1 ) {
      perror( "clock get resolution" );
      return EXIT_FAILURE;
    }
    printf( "Resolution is %ld micro seconds.\n",res.tv_nsec/
             1000);
    return EXIT_SUCCESS;
  }
```

Przykład 1-1 Program testowania dokładności zegara

W testowanym przez Autora przykładzie wynosił 1 ms.

1.1 Funkcje i programowanie timerów

Jedną z najczęściej spotykanych funkcji systemu czasu rzeczywistego jest generowanie zdarzeń które w ustalonym czasie uruchomić mają określone akcje systemu.

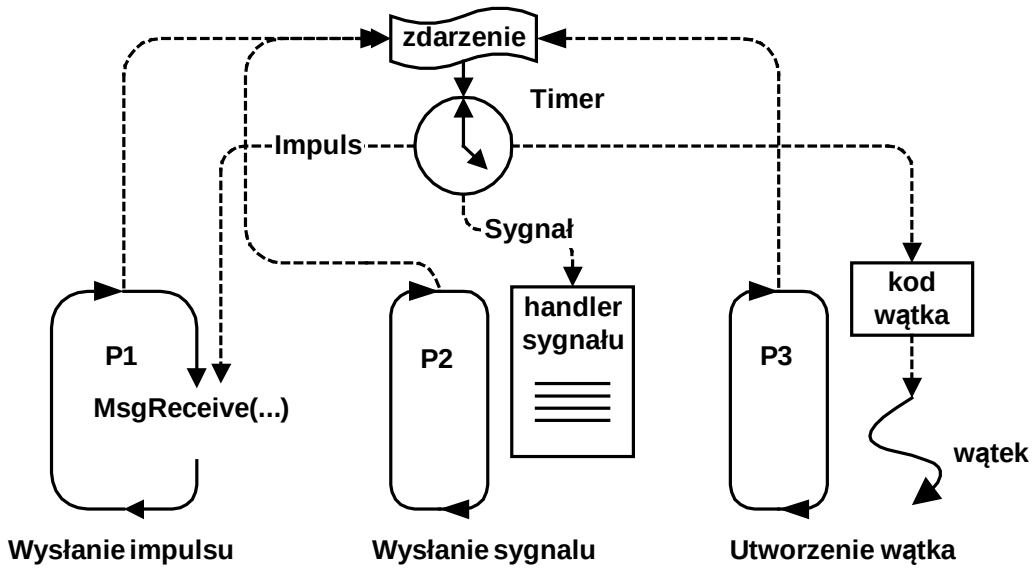
System operacyjny zawiera specjalnie do tego celu utworzone obiekty nazywane timerami (*ang. timers*).

Aby użyć timera należy:

- utworzyć – podaje się specyfikację generowanego zdarzenia
- nastawić – podaje się specyfikację czasu wyzwolenia

W systemie QNX6 Neutrino timery generować mogą następujące typy zdarzeń:

1. Impulsy
2. Sygnały
3. Utworzenie nowego wątku



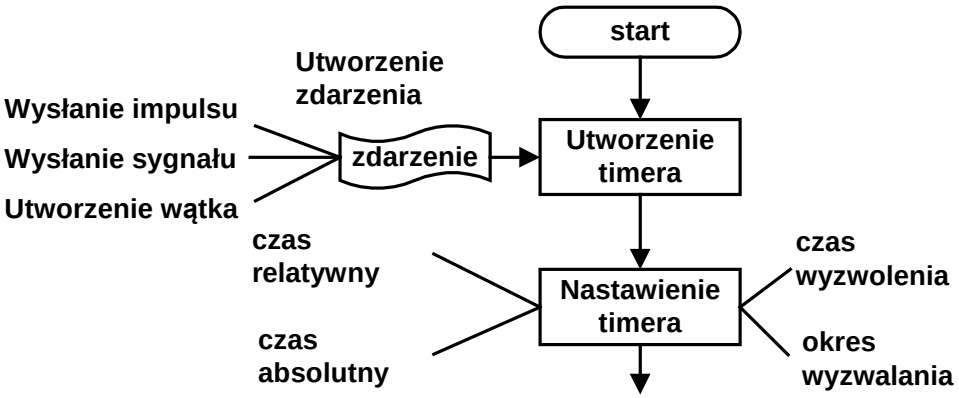
Rys. 1-2 Trzy rodzaje akcji inicjowanych przez timer

Ustawienie timera polega na przekazaniu mu informacji o
- planowanym czasie wyzwolenia,
- sposobie określenia tego czasu
- trybie pracy timera.

Czas można określać w sposób:
- absolutny - czas UTC lub lokalny
- relatywny - przesunięcie czasowe począwszy od chwili bieżącej

Timer może pracować w dwóch trybach:

1. Wyzwolenie jednorazowe (*ang. one shot*)
2. Wyzwalanie cykliczne (*ang. periodical*)



Rys. 1-3 Etapy przygotowania timera do pracy

1.2 Zdarzenia

System QNX6 Neutrino posiada uniwersalny a zarazem jednolity system powiadamiania o zdarzeniach (ang. *event*). Zdarzenie może być:

- impulsem,
- sygnałem
- zdarzeniem które uruchamia wątek.

W zawiadomieniach używa się struktury typu **`sigevent`** zdefiniowanej w pliku nagłówkowym **`<sys/siginfo.h>`**. Znaczenie pól struktury zależy od wartości pola **`sigev_notify`** które określa typ zawiadomienia.

Wartość pola **`sigev_notify`**	Akcja
`SIGEV_PULSE`	Wysłanie impulsu
`SIGEV_SIGNAL`	Wysłanie do procesu sygnału zwykłego
`SIGEV_SIGNAL_CODE`	Wysłanie do procesu sygnału z 8 bitowym kodem
`SIGEV_SIGNAL_THREAD`	Wysłanie do wątku sygnału z 8 bitowym kodem
`SIGEV_INTR`	Używane w przerwaniach
`SIGEV_THREAD`	Utworzenie wątku

Tabela 1-1 Typy zawiadomień w systemie Neutrino

Na poziomie aplikacji używane są zawiadomienia w postaci:

- impulsów,
- sygnałów
- wątków

Wysyłanie sygnałów

Do inicjowania struktury **`event`** można użyć odpowiedniego makra.

Zwykły sygnał:

`SIGEV_SIGNAL_INIT( &event, signal )`

`signal` - numer sygnału.

1.3 Tworzenie i ustawianie timerów

Timer jest obiektem tworzonym przez system operacyjny a jego funkcją jest generowanie zdarzeń w precyzyjnie określonych chwilach czasu.

Aby użyć timera należy wykonać następujące czynności:

1. Zdecydować jaki typ zawiadomień ma generować timer (impulsy, sygnały, uruchomienie wątku) i utworzyć strukturę typu **`sigevent`**.
2. Utworzyć timer.
3. Zdecydować o rodzaju określenia czasu (absolutny lub relatywny).
4. Zdecydować o trybie pracy (timer jednorazowy lub cykliczny)
5. Nastawić go czyli określić tryb pracy i czas zadziałania.

Opis	Funkcja
Utworzenia timera	**`timer_create()`**
Nastawienie timera	**`timer_settime()`**
Uzyskanie ustawień timera	**`timer_gettime()`**
Kasowanie timera	**`timer_delete()`**

Tabela 1-2 Funkcje operujące na timerach

<u>Tworzenie timera</u>

Timer tworzy się za pomocą funkcji **`timer_create()`**.

```
int timer_create(clockid_t clock, struct
sigevent *evn, timer_t *timerid)
```

`clock` Identyfikator zegara użytego do odmierzania czasu obecnie **`CLOCK_REALTIME`**

`evn` Struktura typu sigevent zawierająca specyfikację generowanego zdarzenia.

`timerid` Wskaźnik do struktury zawierającej nowo tworzony timer

Typ powiadomienia określa struktura **sigevent**

- impuls - **SIGEV_PULSE_INIT**
- sygnał - **SIGEV_SIGNAL_INIT, SIGEV_SIGNAL_CODE_INIT, SIGEV_SIGNAL_ THREAD_INIT**
- odblokowanie wątku - **SIGEV_THREAD_INIT**.

Ustawianie timera

Ustawienie timera polega na określeniu:

- sposobu określenia czasu,
- czasu wyzwolenia,
- okresu repetycji.

Do ustawiania timera służy funkcja **timer_settime()**

int timer_settime(timer_t *timerid,int flag, struct itimerspec *val, struct itimerspec *oldval)	
timerid	Identyfikator timera zainicjowany przez funkcję **timer_create**
flag	Flagi specyfikujące sposób określenia czasu 0 – czas relatywny **TIMER_ABSTIME** – czas absolutny
val	Specyfikacja nowego czasu aktywacji
oldval	Specyfikacja poprzedniego czasu aktywacji

```
struct itimerspec {
 struct timespec it_value;        // pierwsza aktywacja
 struct timespec it_interval;     // interwał
}
struct timespec {
  long tv_sec;    // sekundy
  long tv_nsec;   // nanosekundy
}
```

it_value - Czas pierwszego uruchomienie
it_interval - Okres repetycji

it_value	it_interval	Typ zdarzeń
v > 0	x > 0	Cykliczne generowanie zdarzeń co **x** począwszy od **v**
v > 0	0	Jednorazowa generacja zdarzenia w **v**
0	dowolny	Timer zablokowany
x > 0	x > 0	Cykliczne generowanie zdarzeń co **x**

Tabela 1-3 Ustawianie trybu pracy timera

Przykład 1 - timer jednorazowy

```
it_value.tv_sec = 2;
it_value.tv_nsec = 500 000 000;
it_interval.tv_sec = 0
it_interval.tv_nsec = 0;
```

Uruchomi się jednorazowo za 2.5 sekundy od chwili bieżącej.

Przykład 2 - timer cykliczny

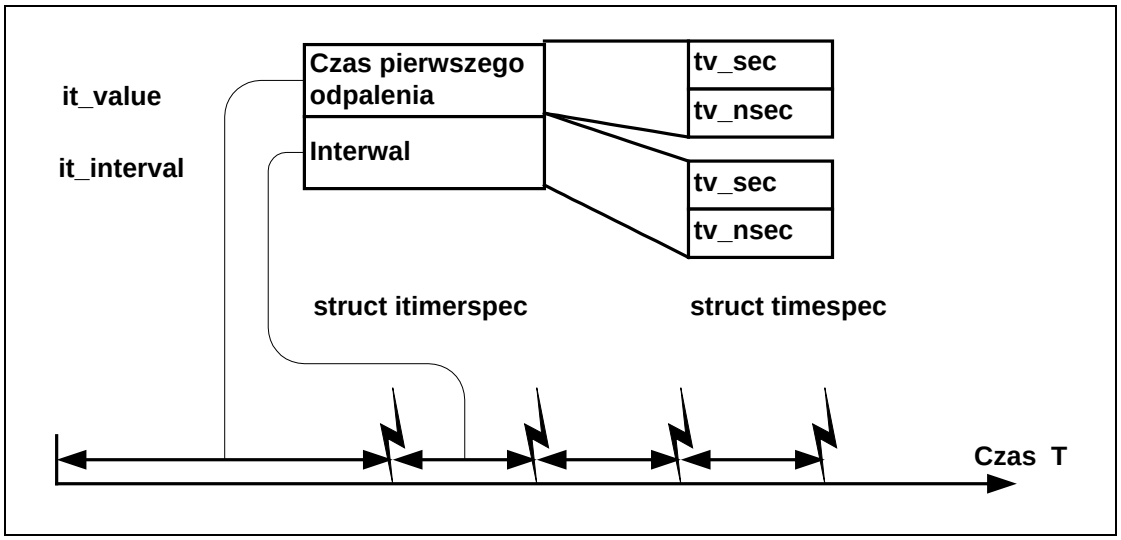
```
it_value.tv_sec = 2;
it_value.tv_nsec = 500 000 000;
it_interval.tv_sec = 1
it_interval.tv_nsec = 0;
```

po upływie 2.5 sekundy będzie generował zdarzenia cyklicznie co 1 sekundę.

Przykład 3 - timer absolutny

```
it_value.tv_sec = 1162378200;
it_value.tv_nsec = 0;
it_interval.tv_sec = 0
it_interval.tv_nsec = 0;
```

1 listopada 2006 roku, godzinie 12.50



Rys. 1-4 Pola struktury **`itimerspec`** dla timera cyklicznego (v = 2.5 sek, x=1sek)

Testowanie timera

Testowanie czasu pozostałego do wyzwolenia timera odbywa się za pomocą funkcji **`timer_gettime()`**.

`int timer_gettime(timer_t *timerid, struct itimerspec *value)`	
`timerid`	Identyfikator timera zainicjowany przez funkcję **`timer_create()`**
`value`	Wskaźnik na strukturę do której skopiowany będzie wynik

Kasowanie timera

Timer kasuje się funkcją **`timer_delete()`**.

`int timer_delete(timer_t *timerid)`	
`timerid`	Identyfikator timera zainicjowany przez funkcję **`timer_create`**

1.4 Przykład – proces cykliczny wykorzystujący sygnały z timera

```
#include <errno.h>
#include <sys/siginfo.h>
#include <signal.h>

int sig_cnt = 0;

void sig_handler( int sig_number )  {
       ++sig_cnt;
}

main(int argc, char *argv[]) {
   int  pid, i ,id,priority;
   struct _msg_info info;
   timer_t timid;
   struct sigevent evn;
   struct itimerspec t;

   // Instalacja funkcji obslugi sygnalu
   signal(SIGUSR1,sig_handler);

   // Utworzenie zdarzenia
   SIGEV_SIGNAL_INIT(&evn,SIGUSR1);

   // Utworzenie timera
   id = timer_create(CLOCK_REALTIME,&evn,&timid);
   if(id < 0) {
         perror("timer");
         exit(-1);
   }

  // Nastawienie timera
  t.it_value.tv_sec = 1;
  t.it_value.tv_nsec = 0;
  t.it_interval.tv_sec= 1;
  t.it_interval.tv_nsec= 0;
  timer_settime(timid,0,&t,NULL);

  // Kod procesu ----------------------------------------
  for(i=0;; i++) {
      sigpause(0);
      printf("Sygnal: %d  \n",sig_cnt);   continue;
  }
}
```

Przykład 1-2 proces cykliczny wykorzystujący sygnały z timera

1.5 Sterowanie sekwencyjne

```
// Ta struktura opisuje pojedynczy krok
typedef  struct  {
    int nr;                // numer kroku
    struct timespec czas;  //  Opoznienie
    int wy;                // Co na wyjsciu
    int we;                // Co na wejsciu
 } krok_t;

 // { krok, {sek, nsek}, wyjscie, wejscie }
 // Tablica prog jest inicjalizowana
 krok_t prog[SIZE]   = { {1, {1,0}, 0x01, 0x01},
                         {2, {2,0}, 0x02, 0x02},
                         {3, {3,0}, 0x04, 0x04},
                         {4, {2,0}, 0x08, 0x08}
                        };

int sig_cnt = 0;
static int base = ADRB;

void sig_handler( int sig_number )  {
       ++sig_cnt;
}

 main(int argc, char *argv[]) {
  int  krok,i =0;
  int id,k,x;
  timer_t timid;
  struct sigevent evn;
  struct itimerspec t;
  ThreadCtl( _NTO_TCTL_IO, 0 );
  base = mmap_device_io(16,ADRB);

  // Instalacja funkcji obslugi sygnalu
  signal(SIGUSR1,sig_handler);

  // Utworzenie zdarzenia
  SIGEV_SIGNAL_INIT(&evn,SIGUSR1);

  // Utworzenie timera
  id = timer_create(CLOCK_REALTIME,&evn,&timid);
   if(id < 0) {
         perror("timer");
         exit(-1);
   }

   krok = 0;
```

```
   for(i=0;i<20; i++) { // Petla glowna
       dout(2,prog[krok].wy);
       // Programowanie timera -------------------
       t.it_value.tv_sec = prog[krok].czas.tv_sec;
       t.it_value.tv_nsec = prog[krok].czas.tv_nsec;
       t.it_interval.tv_sec= 0;
       t.it_interval.tv_nsec= 0;
      // Nastawienie timera
      timer_settime(timid,0,&t,NULL);
      sigpause(0);
      k = 0;
      do {  // Czekamy ma odpowiednie wejscie
           x = dinp(1);
           if( prog[krok].we & x) break;
           usleep(500000);
           k=k+1;
      }  while(k < 10);
      krok = (krok+1) % SIZE;
      printf("Sygnal: %d  krok %d k= %d\n",sig_cnt,krok,k);
  }
  printf("Serwer zakonczony\n");
}
```

Przykład 1-3 Sterowanie sekwencyjne