1. Timery i zdarzenia

1.1 Funkcje i programowanie timerów

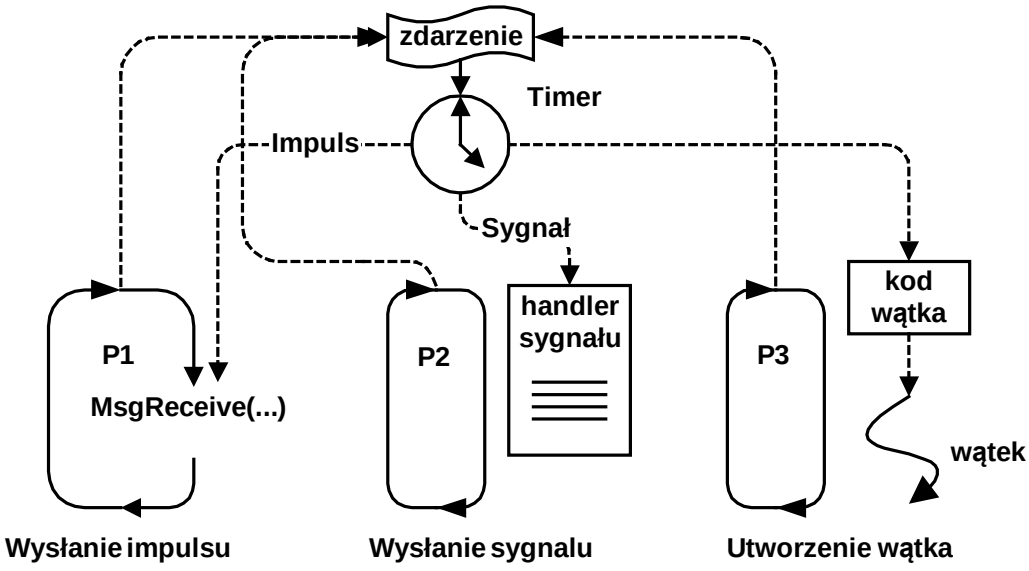
Jedną z najczęściej spotykanych funkcji systemu czasu rzeczywistego jest generowanie zdarzeń które w ustalonym czasie uruchomić mają określone akcje systemu. System operacyjny zawiera specjalnie do tego celu utworzone obiekty nazywane timerami (*ang. timers*).

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. 1-1 Trzy rodzaje akcji inicjowanych przez timer

Ustawienie timera polega na przekazaniu mu informacji o:

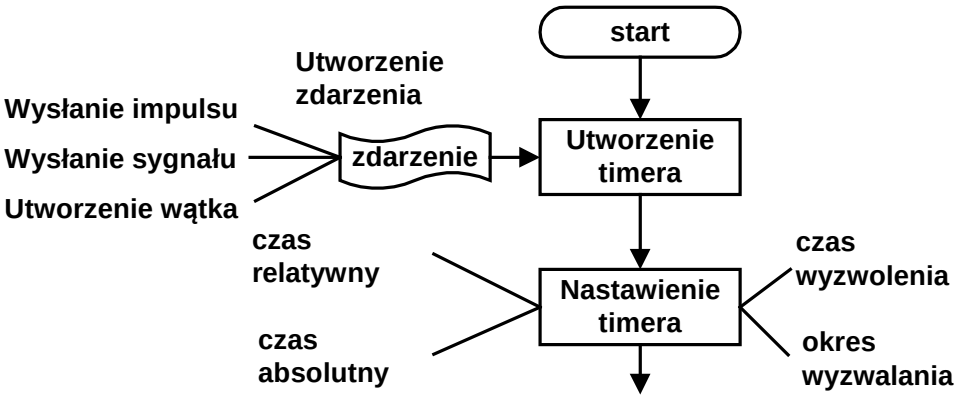
- planowanym czasie wyzwolenia,
- sposobie określenia tego czasu
- trybie pracy timera.

Czas można określać w sposób:

- absolutny - czas UTC lub lokalny
- relatywny - przesunięcie czasowe począwszy od chwili bieżącej

Timer może pracować w dwóch trybach:

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



Rys. 1-2 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.

Wykonywany wątek może otrzymać zdarzenie z jednego z trzech źródeł:

1. Gdy inny wątek wykona funkcję **`MsgDeliverEvent().`**
2. Z procedury obsługi przerwania.
3. Z czasomierza który zakończył odliczanie.

W zawiadomieniach używa się struktury **`sigevent`** zdefiniowanej w pliku nagłówkowym **`<sys/siginfo.h>`**

```
struct sigevent {
  int   sigev_notify;
  union {
   int __sigev_signo;
   int __sigev_coid;
   int __sigev_id;
   void (*__sigev_notify_function)(union sigval);
  }             __sigev_un1;
  union sigval sigev_value;
  union {
     struct {
       short __sigev_code;
       short __sigev_priority;
     } __st;
     pthread_attr_t *__sigev_notify_attributes;
  }             __sigev_un2;
};
```

Listing 1-1 Budowa struktury sigevent

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_UNBLOCK	Odblokowanie przeterminowanego wątku (używane w trybie jądra)
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 impulsów

W przypadku wysyłania impulsów pole **sigev_notify** przyjmuje wartość **SIGEV_PULSE**.

Z każdym impulsem wiąże się:

- kod
- wartość

Impuls wysyłany jest do kanału za pośrednictwem połączenia i zgodnie z pewnym priorytetem.

Atrybuty impulsu zawarte są w polach struktury **sigevent**

Pole	Zawartość
sigev_coid	Pole zawiera identyfikator połączenia CID z kanałem do którego ma być wysłany impuls
sigev_value	32 bitowa wartość związana z impulsem
sigev_code	8 bitowy kod związany z impulsem
sigev_priority	Priorytet impulsu, wartość 0 nie jest dopuszczalna

Tabela 1-2 Pola struktury sigevent gdy wysyłany jest impuls

Makro do inicjowania pól struktury **sigevent** na wysłanie impulsu:

```
SIGEV_PULSE_INIT(&event,coid,priority,code,value )
```

Wysyłanie sygnałów

Wysyłanie sygnałów zachodzi gdy pole **sigev_notify** przyjmie wartość

- **SIGEV_SIGNAL ,**
- **SIGEV_SIGNAL_CODE**
- **SIGEV_SIGNAL_THREAD**

Pole	Zawartość
int sigev_signo	Pole zawiera numer wysyłanego sygnału
short sigev_code	Pole zawiera 8 bitowy kod związany z sygnałem i jest używane z sygnałami typu **SIGEV_SIGNAL_ CODE**

Tabela 1-3 Pola struktury sigevent gdy wysyłany jest sygnał

Do inicjowania struktury można użyć odpowiedniego makra.

Zwykły sygnał:

SIGEV_SIGNAL_INIT(&event, signal)

signal - numer sygnału.

Sygnał z kodem:

SIGEV_SIGNAL_CODE_INIT(&event, signal, value, code)

value - przekazywany do handlera sygnału

code - 8 bitowy kod związany z sygnałem.

Uruchamianie wątków

Gdy pole **sigev_notify** przyjmie wartość **SIGEV_THREAD** to zdarzenie polegało będzie na uruchomieniu wątku.

Pole	Zawartość
sigev_notify_function	Pole zawiera adres funkcji **(void *) func(void *value)**. Z funkcji tej będzie utworzony wątek gdy zajdzie zdarzenie.
sigev_value	Pole zawiera parametr value przekazywany do funkcji **func().**
sigev_notify_attributes	Pole zawiera strukturę z atrybutami wątku który ma być utworzony

Tabela 1-4 Pola struktury sigevent gdy uruchamiany jest wątek

Makro do inicjowania elementów struktury:

SIGEV_THREAD_INIT(&event, func, value, attributes)

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 1-5 Funkcje operujące na timerach

Tworzenie timera

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 1-6 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

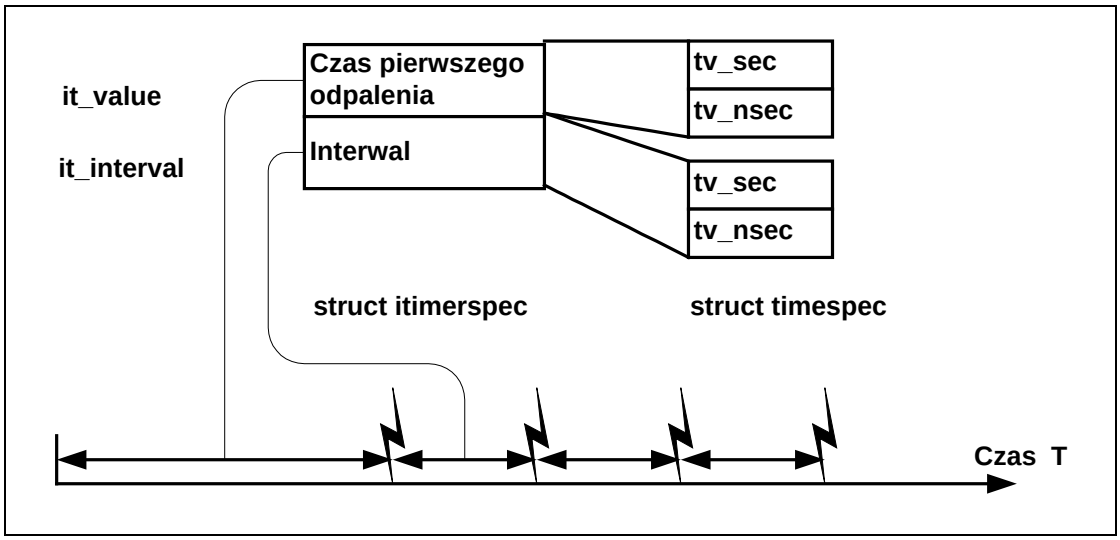
```
it_value.tv_sec = 2;
it_value.tv_nsec = 500 000 000;
it_interval.tv_sec = 1
it_interval.tv_nsec = 0;
```

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. 1-3 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**

Sposób tworzenia timera generującego sygnał i procesu oczekującego na sygnał podaje poniższy przykład.

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

 int sig_cnt = 0;

 // Handler sygnalu
 void sig_handler( int sig_number )  {
       ++sig_cnt;
 }

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

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

   // Utworzenie zdarzenia typu zwykly sygnal
   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);
  printf("Proces startuje \n");
  for(i=0;i<16; i++) {
       pause();
       printf("Sygnal: %d  \n",sig_cnt);   continue;
  }
}
```

Przykład 1-1 Timer generuje sygnały, proces oczekuje na sygnał

W systemach sterowania często zachodzi potrzeba wykonywania określonych czynności w ściśle określonych momentach czasu (często są to czynności cykliczne).
System operacyjny oferuje tu narzędzia:

1. Wykorzystanie opóźnień, funkcje sleep, usleep, delay, nanosleep.
2. Wykorzystanie timerów

`unsigned int sleep( unsigned int seconds )`

`seconds`	Opóźnienie w sekundach

`unsigned int delay( unsigned int duration )`

`duration`	Opóźnienie w milisekundach

`nanosleep(struct timespec* request, struct timespec *remain)`

`request`	Żądane opóźnienie
`remain`	Czas pozostały do zawieszenia procesu

`int usleep( useconds_t useconds );`

`useconds`	Opóźnienie w mikrosekundach

Wykonywanie czynności cyklicznych za pomocą opóźnień ma takie wady jak:

1. Działa tylko wtedy gdy proces jest wykonywany.
2. Precyzja opóźnienia nie może być większa niż częstotliwość szeregowania.
3. Zmienia się wraz z obciążeniem systemu

Część tych wad może być wyeliminowana przez zastosowanie timera.

```
#define ADRB 0x300
#define SIZE     4
#define DIOH    11
#define ADRB 0x300
static int base = ADRB;

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 }
 krok_t prog[SIZE]   = { {1, {1,0}, 0x01, 0x01},
                         {2, {1,0}, 0x02, 0x02},
                         {3, {0,1000000}, 0x04, 0x04},
                         {4, {1,0}, 0x08, 0x08}
                       };

 int sig_cnt = 0;
 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++) {
        out8(base + DIOH,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);
      pause();
      krok = (krok+1) % SIZE;
      printf("Sygnal: %d  krok %d k= %d\n",sig_cnt,krok,k);
  }

}
```

Przykład 1-2 Sterowanie sekwencyjne z użyciem timera

Inne zastosowania timerów:

1. Implementacja przeterminowań
2. Tworzenie wątków