1. Timery i zdarzenia

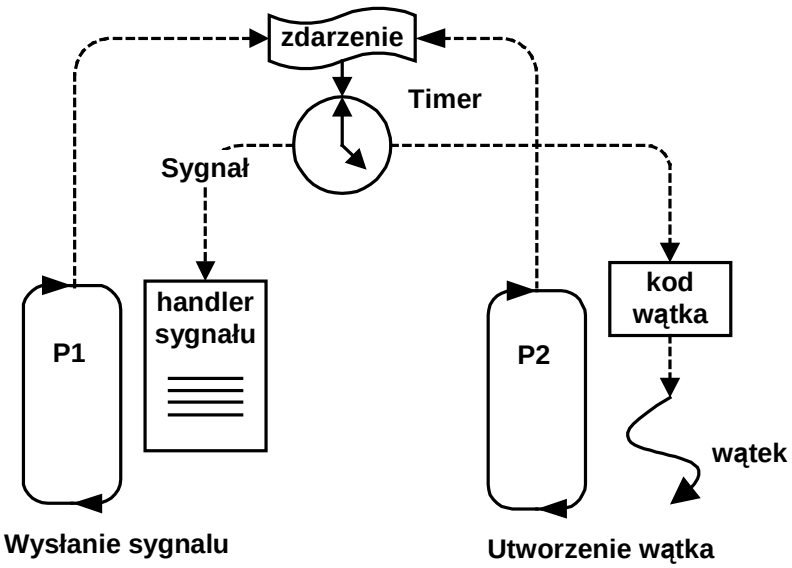
1.1 Funkcje i programowanie timerów

Jedną z najczęściej spotykanych funkcji systemu 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 Linux timery generować mogą następujące typy zdarzeń:

1. Sygnały
2. Utworzenie nowego wątku



Rys. 1-1 Dwa rodzaje akcji inicjowanych przez timer

Ustawienie timera polega na przekazaniu mu informacji o

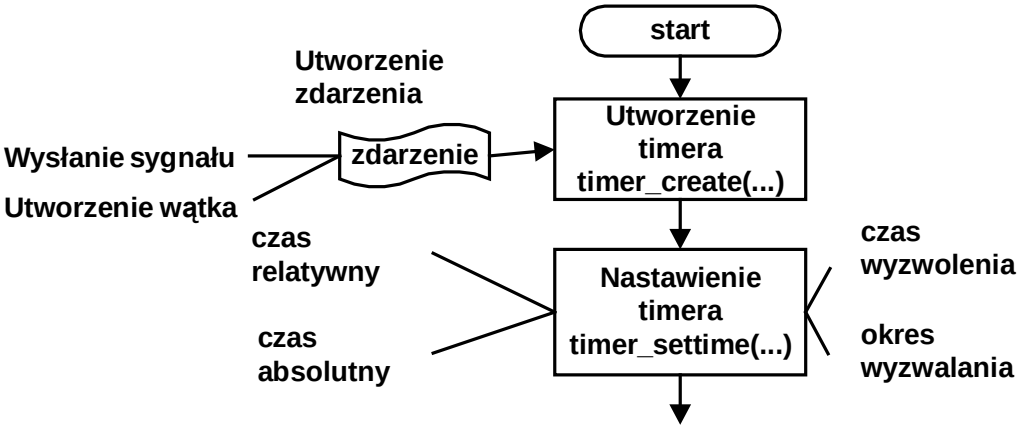
- planowanym czasie wyzwolenia,
- sposobie określenia tego czasu
- trybie pracy timera.

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. 1-2 Etapy przygotowania timera do pracy

1.2 Zdarzenia

System Linux posiada uniwersalny a zarazem jednolity system powiadamiania o zdarzeniach (ang. *event*).

Zdarzenie może być:
- sygnałem
- zdarzeniem które uruchamia wątek.

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

```
union sigval {          /* Data passed with notification */
      int     sival_int;         /* Integer value */
      void   *sival_ptr;         /* Pointer value */
};

struct sigevent {
 int sigev_notify; /* Notification method */
 int sigev_signo;  /* Notification signal */
 union sigval sigev_value;  /* Data passed with
                                       notification */
 void (*sigev_notify_function) (union sigval);
 /* Function used for thread   notification (SIGEV_THREAD) */
 void *sigev_notify_attributes;
 /* Attributes for notification thread (SIGEV_THREAD) */
 pid_t sigev_notify_thread_id;
 /* ID of thread to signal (SIGEV_THREAD_ID) */
 };
```

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_NONE	Brak akcji
SIGEV_SIGNAL	Wysłanie do procesu sygnału zwykłego którego numer zawarty jest w polu sigev_signo
SIGEV_THREAD	Utworzenie wątku, kod wątku zawarty jest w funkcji wskazywanej przez pole sigev_notify_function
SIGEV_THREAD_ID	(Tylko Linux) Działanie takie jak dla SIGEV_SIGNAL ale sygnał kierowany do wątku którego identyfikator zawarty jest w polu sigev_notify_thread_id. Wartość ta zwracana jest przez funkcję **clone** i **gettid**.

Tabela 1-1 Typy zawiadomień w systemie Linux

Na poziomie aplikacji używane są zawiadomienia w postaci:

- sygnałów
- wątków

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 opóźnienia timera	**`timer_getoverrun()`**
Kasowanie timera	**`timer_delete()`**

Tabela 1-2 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`**

- sygnał - **`SIGEV_SIGNAL`**
- odblokowanie wątku - **`SIGEV_THREAD`**

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

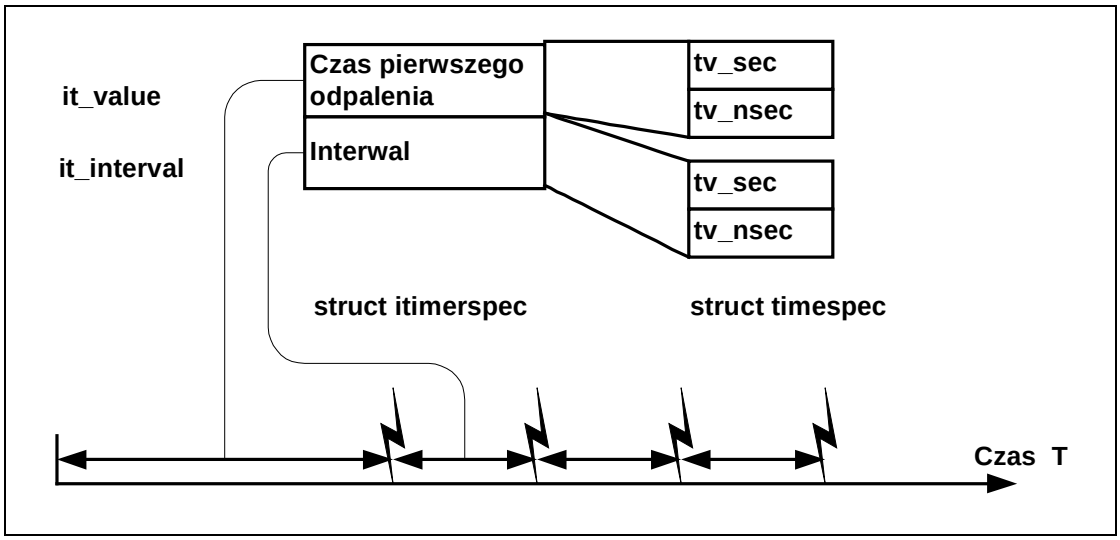
```
it_value.tv_sec = 2;
it_value.tv_nsec = 500 000 000;
it_interval.tv_sec = 1
it_interval.tv_nsec = 0;
```

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. 1-3 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`**

Przykład:

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <signal.h>
#include <time.h>
#define CLOCKID CLOCK_REALTIME
#define SIG SIGRTMIN

#define errExit(msg) do{ perror(msg); exit(EXIT_FAILURE); \
                     } while (0)


static void handler(int sig, siginfo_t *si, void *uc)    {
   printf("Odebrano sygnal %d\n", sig);
   signal(sig, SIG_IGN);
}

int main(int argc, char *argv[]){
   timer_t timerid;
   struct sigevent sev;
   struct itimerspec its;
   long long freq_nanosecs;
   sigset_t mask;
   struct sigaction sa;

   sa.sa_flags = SA_SIGINFO;
   sa.sa_sigaction = handler;
   sigemptyset(&sa.sa_mask);
   if (sigaction(SIG, &sa, NULL) == -1)
               errExit("sigaction");

   // Blokowanie sygnalu SIG
   printf("Blokowanie sygnalu: %d\n", SIG);
   sigemptyset(&mask);
   sigaddset(&mask, SIG);
   if (sigprocmask(SIG_SETMASK, &mask, NULL) == -1)
               errExit("sigprocmask");

   /* Utworzenie timera */
   sev.sigev_notify = SIGEV_SIGNAL;
   sev.sigev_signo = SIG;
   sev.sigev_value.sival_ptr = &timerid;
   if (timer_create(CLOCKID, &sev, &timerid) == -1)
               errExit("timer_create");

   printf("Identyfik. timer jest  %lx\n", (long) timerid);
    /* Start timera */
```

```
    freq_nanosecs = atoll(argv[2]);
    its.it_value.tv_sec = freq_nanosecs / 1000000000;
    its.it_value.tv_nsec = freq_nanosecs % 1000000000;
    its.it_interval.tv_sec = its.it_value.tv_sec;
    its.it_interval.tv_nsec = its.it_value.tv_nsec;

    if (timer_settime(timerid, 0, &its, NULL) == -1)
                errExit("timer_settime");

    /*Blokujemy proces; w międzyczasie timer może być
      wyzwolony wiele razy    */
    printf("Zablokowanie procesu na %d sek\n", atoi(argv[1]));
    sleep(atoi(argv[1]));

     // Odblokowanie sygnalu SIG i oczekiwanie
     printf("Odblokowanie sygnalu %d\n", SIG);
     if (sigprocmask(SIG_UNBLOCK, &mask, NULL) == -1)
               errExit("sigprocmask");
     sleep(atoi(argv[1]));
     exit(EXIT_SUCCESS);
  }
```