

1. Pomiar czasu

1.1 Układy pomiaru czasu w komputerze PC

W typowym komputerze PC do pomiaru czasu przeznaczone są układy:

1. Podtrzymywany zegar czasu rzeczywistego
2. Układ oparty na generatorze kwarcowym, liczniku programowalnym i systemie przerwań.
3. Procesory Pentium posiadają układ liczący cykle procesora

1.1 Zegar czasu rzeczywistego

Komputer PC standardowo posiada podtrzymywany bateryjnie zegar czasu rzeczywistego RTC (*ang. Real Time Clock*) MC146818. Zegar ten pracuje nawet wtedy gdy komputer jest wyłączony.

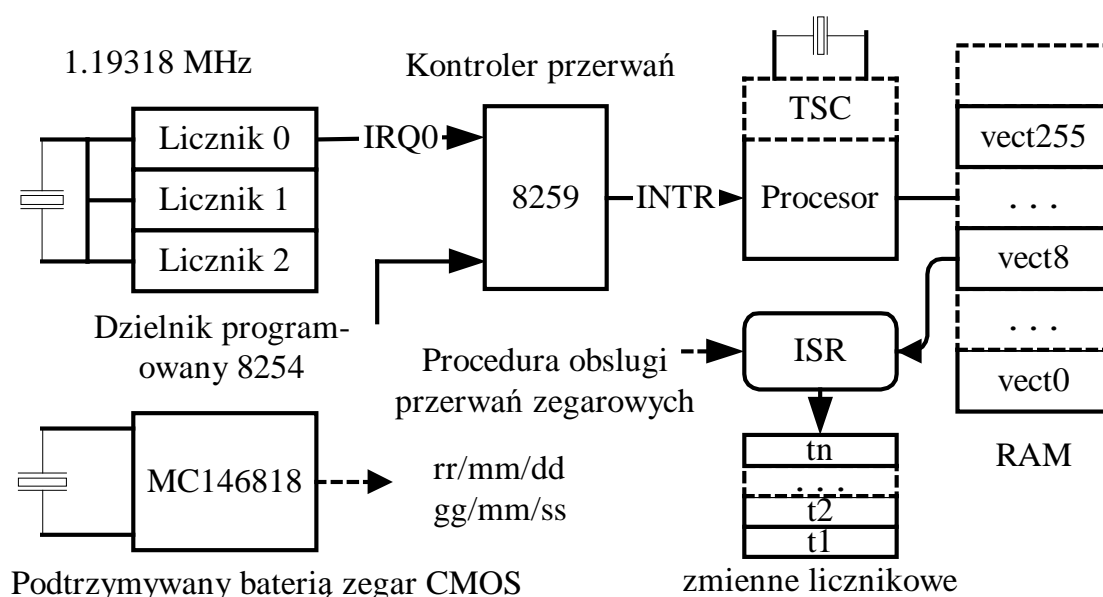
Adres rejestru	Zawartość	Skrót
0	Sekundy	ss
2	Minuty	mm
4	Godziny	gg
6	Dzień tygodnia	tt
7	Dzień miesiąca	dd
8	Miesiąc	mm
9	Dwie ostatnie cyfry roku	rr

Tabela 1-1 Zawartość rejestrów układu MC146818

Czas rzeczywisty uzyskany z zegara RTC przy starcie systemu używany jest do ustawiania czasu systemowego.

1.2 Generator kwarcowy, układ licznikowy i system przerwań

- Pierwotnym źródłem częstotliwości jest generator kwarcowy o częstotliwości 1.19318 MHz.
- Generowana przez ten układ fala prostokątna podawana jest na wejście układu 8254 (trzy programowane liczniki 16 bitowe).
- Licznik 0 dołączony jest do linii wejściowej IRQ0 układu 8259 - głównego kontrolera przerwań systemowych.
- Układ ten z kolei dołączony jest do wejścia INT procesora.



Rys. 1-1 Układy pomiaru czasu w komputerze PC

- Częstotliwość przerwań zegarowych zależna jest od sposobu zaprogramowania licznika 0 układu 8254 (stopnia podziału tego licznika).
- Tak więc im wyższa częstotliwość przerwań zegarowych tym większa dokładność pomiaru czasu w systemie.
- Jednak częstotliwość taka nie może być też zbyt wysoka gdyż obsługa przerwania zegarowego absorbuje określony ułamek mocy procesora.
- Częstotliwość przerwań zegarowych jest kompromisem pomiędzy dokładnością pomiaru czasu a narzutem na obsługę przerwań zegarowych.
- W większości stosowanych obecnie systemów częstotliwość przerwań leży pomiędzy 10 Hz a 1000 Hz

1.3 Licznik cykli procesora Pentium

Procesory rodziny Pentium i P6 posiadają wewnętrzny 64 bitowy licznik cykli procesora nazywany TSC (*ang. Time-Stamp Counter*).

Przy restarcie sprzętowym licznik ten ustawiany jest na zero a następnie zwiększany o 1 co każdy cykl procesora. W procesorze Pentium 1 GHz licznik zwiększany będzie co 1 nanosekundę. Intel gwarantuje że licznik ten nie przepełni się po 10 letniej pracy procesora.

W systemie QNX6 Neutrino rejestr ten odczytać można przy pomocy funkcji `ClockCycles`.

```
uint64_t ClockCycles(void)
```

Funkcja zwraca 64 bitową wartość licznika TSC.

Makroinstrukcja `SYSPAGE_ENTRY` podaje ile cykli procesora przypada na 1 sekundę:

```
SYSPAGE_ENTRY(qtime)->cycles_per_sec
```

```
#include <sys/neutrino.h>
#include <inttypes.h>
#include <stdlib.h>
#include <sys/syspage.h>
// clock.c - program testuje licznik TSC procesora P6
int main( void ){
    uint64_t cps, start, stop, cykli;
    float sec;

    // Ile cykli procesora na sekunde
    cps = SYSPAGE_ENTRY(qtime)->cycles_per_sec;
    printf( "Cykli na sek. %lld \n",cps );

    start=ClockCycles( ); // Start
    printf("Hello\n");      // Jakas instrukcja
    stop=ClockCycles( );   // Stop

    cykli = stop-start;
    printf("Liczba cykli: %lld \n", cykli);
    sec=(float)cykli/cps;
    printf("Liczba sekund %f \n",sec);
    return EXIT_SUCCESS;
}
```

Przykład 1-1 Testowanie czasu wykonania operacji w cyklach

```
Cykli na sek. 2261876400
Hello
Liczba cykli: 4241
Liczba sekund 0.000002
```

Wynik testowania Przykład 1-1 dla procesora Pentium 2.3 GHz

Metoda pomiaru czasu	Rozdzielczość	Uwagi
Podtrzymywany bateryjnie zegar RTC	1 sek	Pracuje przy wyłączonym zasilaniu komputera
Zegar systemowy, licznik, system przerwań	1 ms – 10 ms	Rozdzielczość zależy od częstotliwości przerwań zegarowych
Licznik cykli TSC	1 ns – 100 ns	Występuje nie w każdym systemie

Źródła czasu w komputerze PC

2. Czas systemowy

2.1 Czas nanosekundowy

Aby umożliwić operowanie na krótkich odcinkach czasu (poniżej 1 sekundy) stosuje się format czasu w postaci struktury `timespec`.

Pierwszy element `tv_sec` podaje liczbę sekund która upłynęła Od 1 stycznia 1970 roku.

Drugi element `tv_nsec` podaje liczbę nanosekund które upłynęły od początku bieżącej sekundy.

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

Aby uzyskać dane dotyczące rozdzielczości zegara można użyć funkcji `clock_getres`.

```
int clock_getres(clockid_t id, struct timespec  
*res)
```

Gdzie:

`id` Określenie typu zegara.

`res` Struktura w której określono dokładność zegara

Wraz z wprowadzeniem trybu uśpienia procesora (mogą być usypiane różne elementy procesora) liczenie czasu się komplikuje. System QNX6 rozróżnia następujące tryby zegara.

- **CLOCK_REALTIME** – standardowy czas. Timery bazujące na tym zegarze mogą obudzić procesor nawet gdy jest w stanie uśpienia.
- **CLOCK_SOFTTIME** – Zegar jest aktywny tylko gdy procesor nie jest uśpiony.
- **CLOCK_MONOTONIC** – Wskazanie może się tylko zwiększać, nie może być dostrajany.

```
/* zegar1.c - program testuje rozdzielczosc zegara. */
#include <stdlib.h>
#include <time.h>

int main( void ) {
    struct timespec res;
    int x;
    x = clock_getres( CLOCK_REALTIME, &res);
    if(x != 0) {
        perror( "blad" ); return -1;
    }
    printf("Rozdzielczosc%d us\n",res.tv_nsec / 1000 );
    return EXIT_SUCCESS;
}
```

Przykład 2-1 Uzyskiwanie rozdzielczości zegara systemowego.

Dla komputera wbudowanego Vortex rozdzielczość zegara wynosi 1 ms.

Pytanie czy dokładność zegara można ustawiać? Odpowiedź jest pozytywna ale należy liczyć się z konsekwencjami. Zwiększenie dokładności zegara wiąże się ze zwiększeniem częstotliwości przerwań zegarowych. Jednak im większa częstotliwość przerwań, tym więcej czasu procesora jest zużywana na ich obsługę.

Częstotliwość przerwań zegarowych (a tym samym dokładność zegara) można ustawić za pomocą funkcji ClockPeriod.

```
int ClockPeriod( clockid_t id,
                 const struct _clockperiod * new,
                 struct _clockperiod * old,
                 int reserved )
```

Struktura `_clockperiod` musi zawierać:

<code>unsigned long nsec</code>	– okres zegara w nanosekundach
<code>long fract</code>	– ułamek okresu (nie używane)

Obecnie przyjmuje się:

- Gdy częstotliwość zegara procesora < 40 MHz - 1 mS
- Gdy częstotliwość zegara procesora >= 40 MHz - 10 mS

Bieżący czas systemowy pobiera się za pomocą funkcji `clock_gettime`.

```
int clock_gettime(clockid_t id, struct timespec
*res)
```

Gdzie:

`id` Określenie typu zegara, obecnie tylko `CLOCK_REALTIME`
`res` Struktura w której zawarty będzie aktualny czas systemowy

```
// zegar2.c - program testuje czas wyk. operacji
#include <stdlib.h>
#include <time.h>
#define NANO 1000000000L;

int main( int argc, char *argv[] ) {
    struct timespec start, stop;
    double accum;

    clock_gettime( CLOCK_REALTIME, &start); // Początek
    system( "ls /dev" );                      // Operacja
    clock_gettime( CLOCK_REALTIME, &stop);   // Koniec
    accum = ( stop.tv_sec - start.tv_sec )
           + (double)( stop.tv_nsec - start.tv_nsec )
             / (double)NANO;
    printf( "Czas: %lf\n", accum );
    return 0;
}
```

Przykład 2-2 Pomiar czasu wykonania operacji

Z kolei do ustawiania czasu systemowego używa się podanej niżej funkcji `clock_settime`.

```
int clock_settime(clockid_t id, struct timespec
*res)
```

Gdzie:

`id` Określenie typu zegara, obecnie tylko `CLOCK_REALTIME`
`res` Struktura w której zawarty jest aktualny czas systemowy

Funkcja zwraca 0 gdy sukces a -1 gdy błąd.

2.2 Czas sekundowy

Wiele funkcji systemowych używa starszego formatu `time_t` o rozdzielczości sekundowej – liczba sekund od 1 stycznia 1970 roku. Odpowiada on polu `tv_sec` z typu `timespec`.

```
time_t time(time_t *buf)
```

Gdzie:

buf Bufor do którego czas ma być skopiowany lub NULL

Funkcja zwraca liczbę sekund od 1 stycznia 1970 roku.

2.3 Czas sekundowy w postaci struktury `tm`

Pole struktury	Opis	Zakres
<code>int tm_sec</code>	Sekundy po pełnej minucie	0 – 61
<code>int tm_min</code>	Minuty po pełnej godzinie	0 – 59
<code>int tm_hour</code>	Godziny od północy	0 – 23
<code>int tm_mday</code>	Dzień miesiąca	0 – 31
<code>int tm_mon</code>	Miesiąc	0 - 11
<code>int tm_year</code>	Rok od 1900	
<code>int tm_wday</code>	Dzień w tygodniu , 0 – niedziela	0 – 6
<code>int tm_yday</code>	Dzień roku od 1 stycznia	0 – 365
<code>int tm_isdst</code>	Flaga przesunięcia czasowego	
<code>long int tm_gmtoff</code>	Różnica pomiędzy czasem bieżącym a czasem UTC	
<code>const char * tm_zone</code>	Łańcuch opisu strefy czasowej	

Pola struktury `tm`

Zamiany czasu sekundowego typu `time_t` na strukturę typu `tm` dokonać można używając funkcji `localtime` lub `gmtime`.

```
struct tm *localtime(time_t *time)
```

Gdzie:

Time Czas w sekundach od 1 stycznia 1970 roku

Konwersja ze struktury `tm` na czas sekundowy `time_t`

```
time_t mktime(struct tm *timeprt)
```

Gdzie:

`timeprt` Wskaźnik do struktury typu `tm` zawierającej czas

Funkcja `asctime` przekształca czas wyrażony w postaci struktury `tm` na czas wyrażony w postaci łańcucha tekstowego.

```
char * asctime(struct tm *timep)
```

Gdzie:

`timep` Wskaźnik do struktury typu `tm` zawierającej czas

Funkcja `ctime` przekształca czas wyrażony w postaci czasu sekundowego `time_t` na czas wyrażony w postaci łańcucha tekstowego
- `asctime(localtime(&time))`

```
char * ctime(time_t *time)
```

Gdzie:

`time` Wskaźnik do zmiennej typu `time_t` zawierającej czas sekundowy

```
// czas1.c - testowanie konwersji czasu
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main( void ) {
    time_t t, talarm;
    struct tm ts;
    struct tm *tb;
    // wyprowadzenie daty / czasu jako lancuch
    t = time(NULL);
    printf("Czas biezacy: %s", ctime(&t) );
    // Ustawienie czasu aktywacji na 01/11/2006 12:50
    ts.tm_year = 2006 - 1900;
    ts.tm_mon = 11 - 1;
    ts.tm_mday = 1;
    ts.tm_hour = 12;
    ts.tm_min = 50;
    ts.tm_sec = 0;
    talarm = mktime(&ts);
    while(1) {
        t = time(NULL);
        tb = localtime(&t);
        printf("Rok:%d mies.:%d dzien:%d\n", \
            tb->tm_year + 1900, tb->tm_mon+1, tb->tm_mday);
        printf("Godz:%d min:%d sek:%d\n", \
            tb->tm_hour, tb->tm_min, tb->tm_sec);

        if(t >= talarm) { // Wykonaj planowana czynnosc
            printf("Planowana czynnosc\n");
            break;
        }
        sleep(1);
    }
    return EXIT_SUCCESS;
}
```

Nazwa czasu	Typ danych	Rozdzielczość
Cykle procesora	uint64_t	1 cykl procesora
Nanosekundowy	timespec	1 ns
Sekundowy - liczba int	time_t	1 sek
Sekundowy - struktura	tm	1 sek

Typy czasu w systemie QNX6 Neutrino

3. Opóźnienia

3.1 Opóźnienie sekundowe

W systemach czasu rzeczywistego często zachodzi potrzeba odmierzania odcinków czasu i okresowego wykonywania pewnych zadań.

```
int sleep(int sec)
```

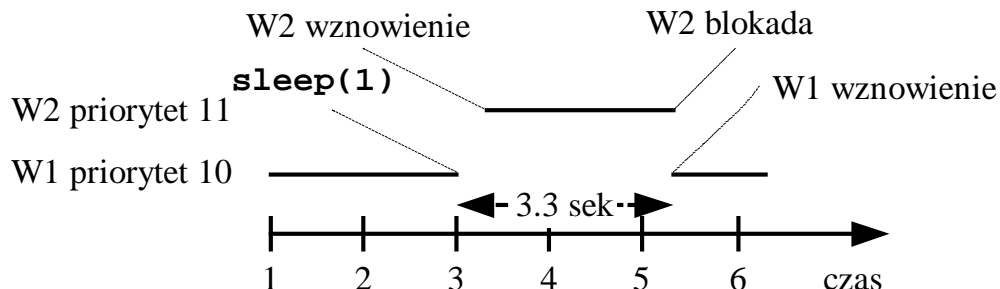
Gdzie:

sec Specyfikacja czasu opóźnienia

Wykonanie funkcji która powoduje zawieszenie wątku na **sec** sekund. Gdy proces otrzyma sygnał to wątek zablokowany na funkcji **sleep** ulegnie odblokowaniu.

Pływanie zegara (ang. *Timing jitter*)

Brak jest gwarancji że po upływie zadanej liczby sekund wątek ulegnie odblokowaniu. Wiadomo jedynie że zostanie on umieszczony w kolejce wątków gotowych. To czy zostanie uruchomiony zależy od obecności innych wątków gotowych o wyższym priorytecie.



Rys. 3-1 Przykład błędu podczas wykonania funkcji `sleep`

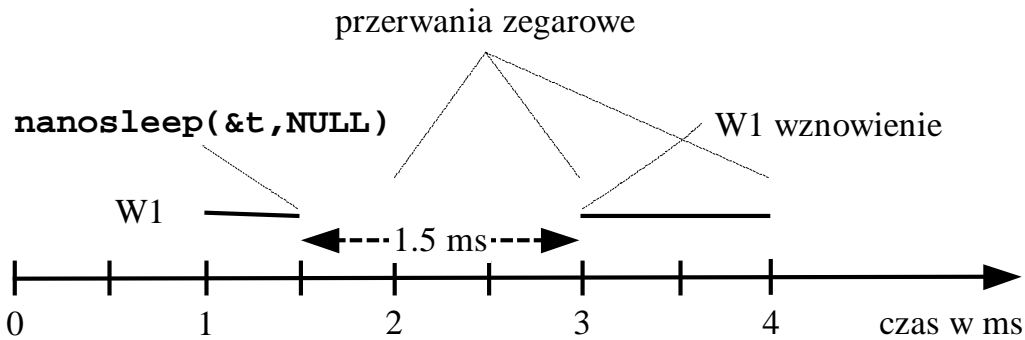
3.2 Opóźnienie nanosekundowe

```
void nanosleep(struct timespec *tp, struct  
timespec *tk)
```

Gdzie:

tp Żądany czas opóźnienia

tk Wskaźnik na strukturę gdzie ma być umieszczony czas pozostały do końca blokady lub NULL



Rys. 3-2 Ilustracja błędu opóźnienia procesu

```
// Program digout2.c - generacja fali prostokątnej T=1 ms
#include <hw/inout.h>
#include <sys/neutrino.h>
#define BASE 300
#define DOUT 3

main(void) {
    unsigned char x = 0;
    struct timespec t;
    t.tv_sec = 0;
    t.tv_nsec = 1000000;
    ThreadCtl(_NTO_TCTL_IO, 0);
    do {
        out8(BASE+DOUT,x);
        if(x==0) x = 1; else x = 0;
        nanosleep(&t,NULL);
    } while(1);
}
```

Generacja impulsu prostokątnego o okresie 1 ms

4. Demon czasowy cron

W systemach komputerowych często zachodzi potrzeba uruchamiania pewnych usług z określonym interwałem czasowym lub też w określonych momentach czasu. Czynności takie mogą być wykonane przez demona **cron**.

Uruchomienie:

```
cron [-d crondir] [-s] [-v] &
```

Gdzie:

crondir – katalog z crontabami o ile różny od **/var/spool/cron**
s – odpytuj co minutę
v – obszerne komunikaty

Demon wykorzystuje tablice (ang. *crontabs*) określające kiedy mają być wykonane czynności. Tablice te zawarte są w katalogu **/var/spool/cron/crontabs**. W poniższym przykładzie widzimy takie tabele dla użytkowników **root** i **ctest**.

```
# ls -l /var/spool/cron/crontabs
total 8
-rw----- 1 ctest crontab 1145 Sep 19 12:09 ctest
-rw----- 1 root crontab 1138 Sep 19 11:50 root
```

Tablice **crontabs** nie powinny być edytowane bezpośrednio ale za pomocą polecenia **crontab**.

```
crontab [ -u user ] file
crontab [ -u user ] [ -i ] { -e | -l | -r }
```

Gdzie:

r – usuń tablicę **crontab**
e – edytuje tablicę **crontab**
l – listuj tablice **crontab**
-u user – nazwa użytkownika
file – nazwa pliku **crontab**

Tablica **crontab** składa się z linii, z których każda definiuje czas wykonania polecenia i zawiera też pewne polecenie. Składnia linii pokazana jest poniżej.

*	*	*	*	*	polecenie do wykonania
					+----- dzień tygodnia (0 - 7),
					+----- miesiąc (1 - 12)
					+----- dzień miesiąca (1 - 31)
					+----- godzina (0 - 23)
					+----- minuta (0 - 59)

Tab. 4-1 Składnia linii tabeli crontab

Dni tygodnia oznaczane są jako: **niedziela=0, poniedziałek=1, wtorek=2, ..., niedziela=7**. Niedziela może być przedstawiona jako 0 lub 7. W tabeli **crontab** dla każdego z parametrów można stosować:

- Wyliczenia, pozycje oddzielone przecinkiem np.
5,10,30 * * * * polecenie
co znaczy wykonaj polecenie w minucie 5,10,30.
- Przedziały, pozycje oddzielone znakiem -, np.
0,30 7-15 * * * polecenie
co znaczy wykonaj polecenie w minucie 0,30 w godzinach od 7 do 15
- Interwały, po pozycji znak */interwał, np.
***/10 * * * * polecenie**
co znaczy wykonaj polecenie co 10 minut.
- Gdy w linii ma być zdefiniowane kilka zadań należy rozdzielić je średnikiem, np. jak poniżej.
0,30 * * * * polecenie1; polecenie2

W przypadku poleceń uruchamianych przez demona **cron** występuje problem z określeniem środowiska i **standard input/output**. Jako że zadania uruchamiane przez **cron** nie mają standardowego wyjścia, nie powinny generować żadnych komunikatów. Osiągnięte jest to poprzez podany niżej sposób.

```
* * * polecenie > /dev/null 2>&1
```

Pierwsza część czyli **> /dev/null** znaczy że standardowe wyjście należy skierować na urządzenie wirtualne **/dev/null** gdzie znaki są ignorowane. Druga część wyrażenia **2>&1** znaczy że standardowe wyjście błędów (ma uchwyt 2) będzie skierowane na standardowe wejście (ma uchwyt 1). Rezultatem będzie brak jakichkolwiek komunikatów. W tabeli **crontab** należy określić środowisko a przykład tabeli podano poniżej.

```
# crontab -l
SHELL=/bin/bash
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin
# Zapisuj co minute i co godzinie w mies. 10 tekst do rejestru
# /var/log/user.log
* * * 10 *    logger -p user.info To jest wpis z crona
# Zapisuj co minute i co godzinie obciazenie systemu do pliku
/tmp/obciaz.txt
* * * * *    uptime >> /tmp/obciaz.txt
```

Przykład 4-1 Przykład tablicy crontab