CZAS

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.

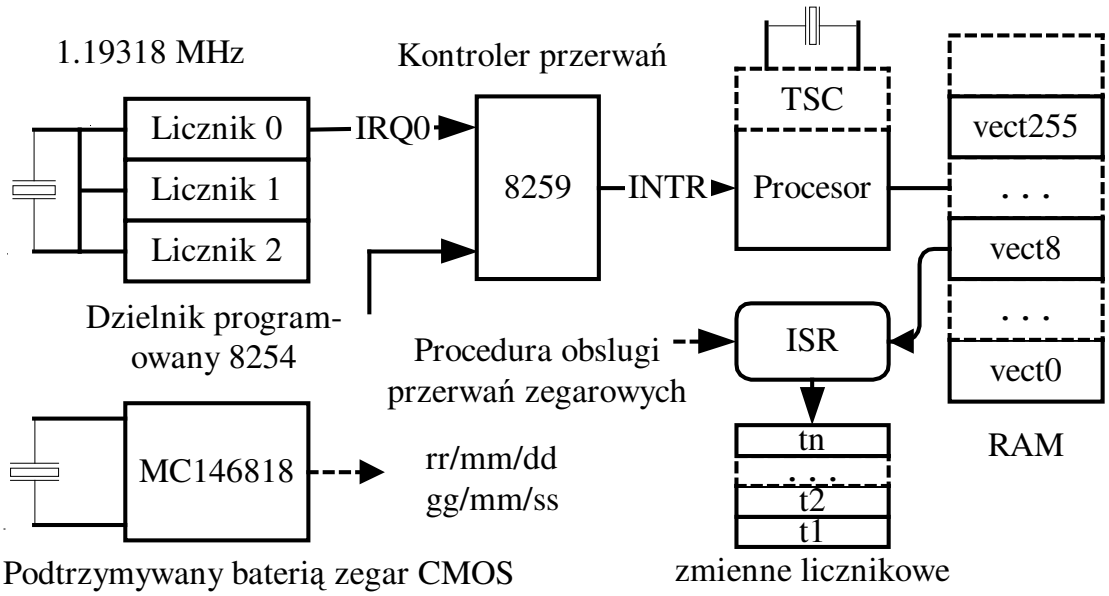
Adres rejestru	Zawartość	Skrót
0	Sekundy	**ss**
2	Minuty	**mm**
4	Godziny	**gg**
6	Dzień tygodnia	**tt**
7	Dzień miesiąca	**dd**
8	Miesiąc	**mm**
9	Dwie ostatnie cyfry roku	**rr**

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 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 bieżacej 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, obecnie tylko **CLOCK_REALTIME**
res Struktura w której określono dokładność zegara

```
/* 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.

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

Pola struktury **tm**

Zamiany czasu sekundowego typu **time_t** na strukture 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.

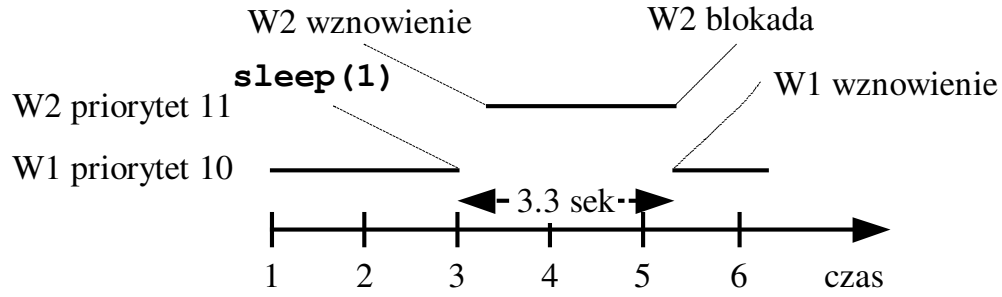
```
int sleep(int sec)
```

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.

Brak gwarancji że po upływie zadanej liczby sekund watek 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.



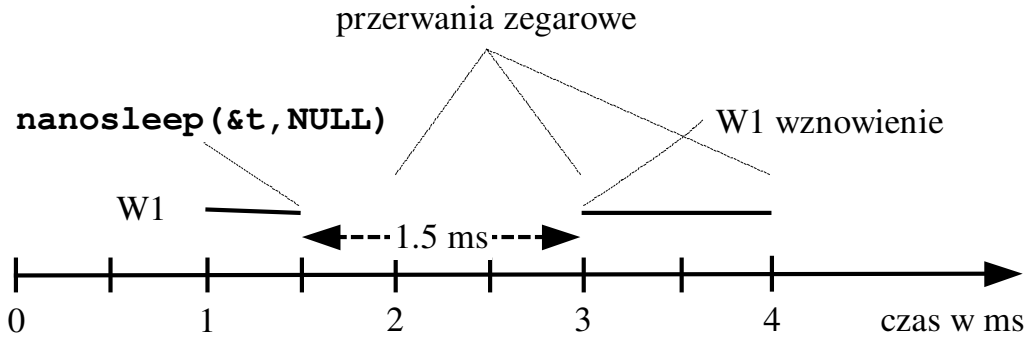
3.2 Opóźnienie nanosekundowe

```
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



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. Uruchamianie procesów w zadanym czasie – serwer cron

W systemach sterowania niejednokrotnie istnieje potrzeba automatycznego uruchomienia procesu w okreslonym momencie czasowym. Taką mozliwość zapewnia serwer czasu **cron**.

Proces cron uruchamia się automatycznie przy starcie systemu lub poprzez wpis w pliku inicjalizacyjnym systemu: **/etc/rc.d/rc.local**

cron –d crondir –v &

Proces ten ma swoje pliki konfiguracyjne w domyślnym katalogu:

/var/spool/cron

Domyślny katalog może być zmieniony na inny poprzez opcję **-d crondir**
-v – uruchomienie z opcją diagnostyczną

/var/spool/cron/cron.allow	Plik zawierający nazwy użytkowników którzy mogą korzystać z usługi. Domyslnie wszyscy mogą (gdy nie ma pliku)
/var/spool/cron/cron.deny	Plik zawierający nazwy użytkowników którzy nie mogą korzystać z usługi.
/var/spool/cron/crontabs	Pliki konfiguracyjne użytkowników

Rys. 4-1 Katalogi używane przez serwer czasu cron

Specyfikacji czasu wykonania i polecenia dokonuje się umieszczając wpisy w tablicy crontab. Wykonuje się to za pomocą polecenia:

crontab [-d cron_dir] [-e|-l|-r] [-u user]

-d cron_dir	Katalog w którym umieszczono plik crontab
-e	Edycja pliku crontab. Edytor można ustawić poprzez zmienną środowiska EDITOR
-l	Listowanie pliku crontab
-r	Usunięcie pliku crontab
-u user	Nazwa uzytkownika

Gdy użyjemy opcji –e włącza się domyślny edytor i należy dokonać edycji poszczególnych linii odpowiadających poleceniom.

```
*     *     *     *     *  polecenie
|     |     |     |     |
|     |     |     |     +----- dzień tygodnia (0 - 7)
|     |     |     +------- miesiąc (1 - 12)
|     |     |
|     |     +--------- dzień miesiąca (1 - 31)
|     |
|     +----------- godzina (0 - 23)
|
+------------- minuta (0 - 59)
```

Rys. 4-2 Zawartość linii pliku crontab

Uwagi:

- Dni tygodnia oznaczane są nastepująco: 0 – niedziela, 1 – poniedziałek, ..., 7 – niedziela. Niedziela może być oznaczona jako 0 lub 7
- W crontabie należy podawać pełne ścieżki do poleceń lub ustawić odpowiednią wartość zmiennej PATH.
- Dzień wykonania komendy można ustalić na podając dzień miesiąca lub dzień tygodnia. Jeśli oba pola są ustawione, to komenda wykona się zarówno w ustawiony dzień miesiąca, jak i w ustawiony dzień tygodnia.
- Wartości liczbowe możemy zapisywać w różnych formatach:
    - 1-4 - wartości 1,2,3,4
    - 1,2,6 - wartości kolejno 1,2,6
- Standardowe wyjście z uruchamianych programów kierowane jest w postaci email na skrzynkę pocztową właściciela. Aby tego uniknąć należy przekierować standardowe wyjście na /dev/null , /dev/con lub podobnie

Przykłady:

```
0 0 1,16 * 1 polecenie
```

Uruchomienie o godz; 00:00 dnia 1 i 16 w poniedziałek każdego miesiąca

```
 1,21,41 * * * * date > /dev/con
```

Co 20 minut uruchamiane polecenie date i czas wyprowadzany na konsolę.

Serwer czasu cron może być użyty do uruchamiania programów wykonujących sterowanie ale rozdzielczość czasu 1 sekunda nie zawsze jest wystarczająca.