Pomiar czasu

Metoda pomiaru powinna być dostosowana do mierzonych przedziałów czasu tak, aby uzyskać dokładność przynajmniej rzędu pojedynczych procentów. Najprostszym wariantem jest użycie funkcji *clock()*, tak jak w pokazanym poniżej przykładzie:

```
#include <windows.h>
#include <iostream>
#include <ctime>

using namespace std;

//---------------------------------------------------------------------------
void main(void)
{
clock_t start, elapsed;

  start = clock();
  Sleep(1000);                  // kod ktorego czas wykonania mierzymy
  elapsed = clock() - start;

  cout << "Time [ms] = " << (1000 * elapsed) / CLOCKS_PER_SEC << endl;

  system("PAUSE");
}
```

Funkcja *clock()* ma jednak dość małą rozdzielczość, rzędu kilkunastu milisekund. Aby błąd pomiaru był akceptowalny, można w ten sposób mierzyć tylko czasy rzędu sekund. Dokładniejszy pomiar (rozdzielczość mikrosekundowa) można uzyskać wykorzystując funkcje *QueryPerformanceCounter()*, ilustruje to poniższy przykład:

```
#include <windows.h>
#include <iostream>
#include <iomanip>

using namespace std;

//---------------------------------------------------------------------------
long long int read_QPC()
{
LARGE_INTEGER count;

  QueryPerformanceCounter(&count);
  return((long long int)count.QuadPart);
}

//---------------------------------------------------------------------------
void main(void)
{
long long int frequency, start, elapsed;

  QueryPerformanceFrequency((LARGE_INTEGER *)&frequency);

  start = read_QPC();
  Sleep(1000);                  // kod ktorego czas wykonania mierzymy
  elapsed = read_QPC() - start;

  cout << "Time [s]  = " << fixed << setprecision(3) << (float)elapsed /
          frequency << endl;
  cout << "Time [ms] = " << setprecision(0) << (1000.0 * elapsed) /
          frequency << endl;
  cout << "Time [us] = " << setprecision(0) << (1000000.0 * elapsed) /
          frequency << endl << endl;

  system("PAUSE");
}
```

Funkcja *QueryPerformanceCounter()* (wprowadzona od Windows 2000 i Windows XP) zwraca tylko stan odpowiedniego licznika. Aby uzyskać pomiar w jednostkach czasu potrzebna jest jeszcze częstotliwość impulsów tego licznika, którą można odczytać wywołując funkcję *QueryPerformanceFrequency()*. Końcowy rezultat uzyskuje się przez podzielenie odmierzonej liczby impulsów licznika przez częstotliwość.

Uruchamiając kod pokazany w przykładzie można zauważyć, że uzyskany czas może różnić się od spodziewanego 1000 ms w granicach kilkunastu milisekund. Nie wynika to jednak z metody pomiaru tylko z niedokładności odmierzania czasu przez funkcję *Sleep()*, która jest podobna jak dla funkcji *clock()*.

W przypadku problemów ze stabilnością pomiarów można spróbować użyć bardziej rozbudowanego wariantu funkcji *read_QPC()*:

```
//------------------------------------------------------------------------
// On a multiprocessor computer, it should not matter which processor
// is called. However, you can get different results on different
// processors due to bugs in the basic input/output system (BIOS)
// or the hardware abstraction layer (HAL). To specify processor
// affinity for a thread, use the SetThreadAffinityMask function.
//------------------------------------------------------------------------
long long int read_QPC()
{
LARGE_INTEGER count;

  DWORD_PTR oldmask = SetThreadAffinityMask(GetCurrentThread(), 0);
  QueryPerformanceCounter(&count);
  SetThreadAffinityMask(GetCurrentThread(), oldmask);

  return((long long int)count.QuadPart);
}
```