# Obliczenia „w tle” bez współbieżności

```
while (TRUE)
        if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
        {
                // pobrany komunikat z kolejki

                if (msg.message == WM_QUIT) // zakończ aplikację
                        break;

                TranslateMessage(&msg);
                DispatchMessage(&msg);
        }
                else
        {
                // kolejka komunikatów aktualnie pusta

                „ZróbCośwTle();”
        }
```

# Utworzenie nowego wątku

unsigned long ***_beginthread*** *(*start_adress, stack_size, arg_list)

**unsigned stack_size -** rozmiar stosu (0 – taki sam jak dla wątku aktywnego)
**void *arg_list -** parametr dla nowego wątku do wykorzystania przez programistę np. wskaźnik do struktury danych.
Funkcja zwraca **uchwyt** do nowo utworzonego wątku

```
void ThreadProc(void *arg_list)
{
                    // Kod nowego wątku

      return;       // automatycznie wywołuje _endthread
}
```

**arg_list -** parametr przekazany do **_beginthread.**

# Sekcje krytyczne

Zapewniają proste rozwiązanie problemu wzajemnego wykluczania (ang. *mutual exclusion*).

**CRITICAL_SECTION** cs;

***InitializeCriticalSection***(&cs); - utworzenie sekcji krytycznej.
***DeleteCriticalSection***(&cs); - usunięcie sekcji krytycznej.

***EnterCriticalSection***(&cs) - wątek wchodzi w posiadanie sekcji krytycznej cs. Od tego momentu próba wejścia do sekcji krytycznej przez inny wątek spowoduje wstrzymanie tego wątku do momentu zwolnienia sekcji krytycznej przez wątek który ją posiada.

***LeaveCriticalSection***(&cs) - wątek opuszcza sekcję krytyczną cs. Od tego momentu inne wątki mogą wejść do sekcji krytycznej.

# Zdarzenia (ang. event)

Obiekt zdarzenia może znajdować się w jednym z dwóch stanów: zasygnalizowanym (ang. signaled) lub niezasygnalizowanym (non-signaled). Tworzony jest przy pomocy funkcji

**HANDLE *CreateEvent*(NULL, bManualReset , bInitial, NULL),**

zwracającej uchwyt do utworzonego obiektu zdarzenia. Dla zdarzeń stosowanych do synchronizacji wątków w obrębie jednego procesu pierwszy i ostatni parametr powinny być równe **NULL**.
**bInitial –** początkowy stan zdarzenia (TRUE – zasygnalizowane)
**bManualReset – FALSE** system automatycznie zresetuje (przestawi w stan niezasygnalizowany) zdarzenie po zwolnieniu pojedynczego wątku czekającego na zasygnalizowanie zdarzenia.

Uniwersalna funkcja ***CloseHandle***(hEvent) zamyka obiekt zdarzenia.

***SetEvent***(hEvent) oraz ***ResetEvent***(hEvent) sygnalizują oraz resetują obiekt zdarzenia hEvent.

Uniwersalna funkcja ***WaitForSingleObject***(hEvent, dwTimeOut) wstrzymuje wołający wątek do czasu zasygnalizowania zdarzenia, ale co najwyżej na **dwTimeOut** milisekund. **dwTimeOut** może być równe **INFINITE** (brak ograniczenia czasowego). Jeżeli zdarzenie było utworzone z parametrem **bManualReset = FALSE,** to po wznowieniu wątku zostanie automatycznie zresetowane.

***PulseEvent***(hEvent) ustawia obiekt zdarzenia w stan zasygnalizowany, zwalnia wszystkie czekające wątki (w przypadku zdarzenia dla którego **bManualReset = FALSE** tylko **jeden** watek), po czym z powrotem ustawia stan zdarzenia na niezasygnlizowany.

# Przykład

```
#include <process.h>

const int MaxIter=1000000;
volatile int Counter; // zmienna ulotna, nie zostanie przechowana w rejestrze
CRITICAL_SECTION CS;
HANDLE hEvent;

void LoopIncrement()
{
        for(int i=0;i<MaxIter;i++)
        {
        EnterCriticalSection(&CS);
        int Temp=Counter;
        for(int j=0;j<100;j++)
                ;
        Counter=Temp+1;
        LeaveCriticalSection(&CS);
        }
}
void LoopDecrement()
{
        for(int i=0;i<MaxIter;i++)
        {
        EnterCriticalSection(&CS);
        int Temp=Counter;
        for(int j=0;j<100;j++)
                ;
        Counter=Temp-1;
        LeaveCriticalSection(&CS);
        }
}
void BackgroundThread(void *arg)
{
        LoopIncrement();
        SetEvent(hEvent);
        _endthread();
}
void MainThread()
{
        InitializeCriticalSection(&CS);
        hEvent=CreateEvent(NULL,FALSE,FALSE,NULL);
        _beginthread(BackgroundThread,0,NULL);
        LoopDecrement();
        WaitForSingleObject(hEvent,INFINITE);
        CloseHandle(hEvent);
        DeleteCriticalSection(&CS);
}
```