Wielowątkowość w COM

Wielowątkowość w COM

Współczesne systemy operacyjne pozwalają na współbieżną pracę wielu wątków. Wielowątkowe aplikacje wymagają szczególnie wnikliwej analizy ich działania ze względu na dostęp do współdzielonych zasobów. Problem ten obejmuje również dostęp do komponentów COM.

System COM pozwala projektantowi komponentu wybrać różne modele współpracy z klientem w zależności od przewidywanych warunków pracy komponentu i wymagań co do szybkości jego pracy.

Wielowątkowość w COM

Zdefiniowane są następujące modele współpracy wielowątkowej dla komponentu:

- **Brak (*single*, *none*) - wszystkie instancje komponentu obsługiwane są ze wspólną kolejką zdarzeń związaną z wątkiem, który pierwszy zainicjalizował komponent. Tylko ten wątek ma bezpośredni dostęp do obiektu, pozostałe wątki wymagają *marshalingu* przez kod proxy/stub. Zakończenie wątku związanego z komponentem niszczy wszystkie instancje komponentu. Zaleca się więc aby pierwsza instancja komponentu utworzona była w głównym wątku klienta. Jeśli komponent współdzieli dane pomiędzy instancje, to ten model zapewnia dla nich synchronizację dostępu.**

Wielowątkowość w COM

- **Apartment - każda instancja komponentu ma własną kolejkę zdarzeń. Pozwala to na równoległą pracę różnych wątków z różnymi instancjami komponentu. Wymagane jest zastosowanie synchronizacji do danych współdzielonych przez instancje. Dostęp do danych prywatnych instancji jest synchronizowany przez kolejkę zdarzeń.**

Wielowątkowość w COM

- **Free - metody obiektu wywoływane są bezpośrednio przez vtable komponentu. Takie podejście zapewnia najszybszy dostęp do obiektu i równoległą obsługę różnych wątków przez tą samą instancję komponentu. Wymagane jest zastosowanie metod synchronizacji (*mutex*, *semaphore*, *critical section*) podczas dostępu do danych komponentu.**
- **Both - umożliwia działanie komponentu zarówno w trybie Free jak i Apartment w zależności od wymagań klienta**

Wielowątkowość w COM

Klient podczas inicjalizacji komponentu również narzuca własny model współpracy podczas wywołania `CoInitialize()` i `CoInitializeEx()`:

- **`COINIT_APARTMENTTHREADED`**
- **`COINIT_MULTITHREADED`**

W trybie `COINIT_APARTMENTTHREADED` wątek klienta tworzący instancję komponentu posiada własny podsystem COM (kolejkę zdarzeń). Żaden inny wątek nie ma bezpośredniego dostępu do tego podsystemu. Użycie wskaźnika do obiektu COM z innego wątka wymaga zastosowania marshalingu do jego przekazania. Dostęp do instancji z innych wątków odbywa się poprzez proxy.

Wielowątkowość w COM

Wątki zainicjalizowane w trybie `COINIT_MULTITHREADED` posiadają wspólny podsystem COM. W ramach tego podsystemu wskaźniki do interfejsów mogą być przekazywane bezpośrednio. Dostęp do interfejsu odbywa się bezpośrednio.

W zależności od kombinacji modelu współpracy zażądanego przez klienta i modelu wspieranego przez komponent serwer COM odpowiednio dostosowuje sposób wymiany danych pomiędzy klientem i komponentem.

Wielowątkowość w COM

Model		Sposób komunikacji wątku ze swoją instancją komponentu
komponentu	klienta	
None Single	AP	Główny wątek ma bezpośredni dostęp do komponentu, pozostałe wątki używają proxy.
	MT	Dostęp poprzez proxy. Gdy klient uruchomi komponent w trybie MT, tworzony jest pomocniczy wątek obsługujący odwołania komponentu do klienta.
Apartment	AP	Każda instancja obiektu ma własną kolejkę wywołań, dostęp do obiektu utworzonego w bieżącym wątku następuje bezpośrednio.
	MT	Dostęp poprzez proxy. Gdy klient uruchomi komponent w trybie MT, tworzony jest pomocniczy wątek obsługujący odwołania komponentu do klienta.
Free	AP	Dostęp poprzez proxy.
	MT	Dostęp bezpośredni.
Both	AP	Dostęp bezpośredni.
	MT	Dostęp bezpośredni.

Wielowątkowość w COM

UWAGA!
Możliwość bezpośredniego dostępu do wspólnej instancji komponentu z różnych wątków warunkowana może być tym, jaki model współpracy wybrał wątek inicjalizujący tą instancję. Inicjalizacja modelem `COINIT_APARTMENTTHREADED` wymusza użycie proxy przy dostępie z innych wątków (komunikacja pomiędzy dwoma podsystemami COM).

Jeśli więc przewiduje się, że klienci będą używali wielowątkowego dostępu do obiektu, chcąc uniknąć użycia proxy, należy wszystkie wątki zainicjalizować w trybie `COINIT_MULTITHREADED`.

Dla aplikacji, w których równoległe przetwarzanie wywołań przez komponent, może znacząco poprawić wydajność działania, należy stosować model *Free* lub *Both*.

Wielowątkowość w COM

```
HRESULT InterfaceToStream(
        REFIID riid, IUnknown* pUnknown, IStream** pStream)
{

        CreateStreamOnHGlobal(NULL, TRUE, pStream);
        return CoMarshalInterface(*pStream, riid, pUnknown,
            MSHCTX_INPROC, NULL, MSHLFLAGS_NORMAL);
}

HRESULT StreamToInterface(
        IStream* pStream, REFIID riid, void** ppv)
{
        HRESULT hr=CoUnmarshalInterface(pStream, riid, ppv);
        pStream->Release();
        return hr;
}
```

Asynchroniczne wywoływanie metod

Począwszy od Windows 2000 system COM wspiera asynchroniczne wywoływanie metod. Implementacja tej właściwości wymaga jedynie niewielkiej modyfikacji kodu IDL:

```
[
  object,
  uuid(3C7CB3FB-9AEE-4BB3-9FDD-CD81CF1902BC),
  async_uuid(3C7CB3FB-9AEE-4BB3-9FDD-CD81CF1902BD),
  helpstring("ICalculator Interface"),
  pointer_default(unique)
]
interface ICalculator : IUnknown
{
  HRESULT Compute([in] BSTR expression,
                  [in, out] float* parameters[10]);
                  [out] float* result);
};
```

Asynchroniczne wywoływanie metod

Użycie atrybutu `async_uuid` dodaje do implementacji komponentu interfejs `ICallFactory`. Służy on do zainicjowania asynchronicznego odpowiednika definiowanego interfejsu. Generowany jest jednocześnie bliźniaczy interfejs o nazwie `AsyncIOryginalName` (`AsyncICalculator`).

```
[
  uuid(3C7CB3FB-9AEE-4BB3-9FDD-CD81CF1902BD),
]
interface AsyncICalculator : IUnknown
{
  HRESULT Begin_Compute([in] BSTR expression,
                        [in] float parameters[10]);
  HRESULT End_Compute  ([out] float parameters[10]);
                        [out] float* result);
};
```

Dostęp do asynchronicznego interfejsu uzyskuje się poprzez wywołanie metody `CreateCall()` interfejsu `ICallFactory`.

Asynchroniczne wywoływanie metod

Do sprawdzania statusu operacji służy obiekt `ISynchronize` dostępny jako jeden z interfejsów asynchronicznego komponentu:

```
ISynchronize* pSynchronize;
pAsyncCalculator->QueryInterface(IID_ISynchronize,
                                 (void**)&pSynchronize);
```

Asynchroniczne wywoływanie metod

```
interface ISynchronize : IUnknown
{
    HRESULT Wait([in] DWORD dwFlags, [in] DWORD dwMilliseconds);
    HRESULT Signal();
    HRESULT Reset();
}
```

`Wait()` oczekuje przez zdefiniowany czas na zgłoszenie sygnału do obiektu. Jeśli nie nadszedł sygnał, zwracany jest wynik `RPC_S_CALLPENDING`. Po nadejściu sygnału zwracany jest `S_OK`. Parametr `dwFlags`, pozwala na określenie zachowania funkcji jeśli więcej niż jeden uchwyt do obiektu synchronizacji jest skojarzony z `ISynchronize`:

- **0 - funkcja zwraca S_OK, jeśli jeden z obiektów otrzymał sygnał**
- **`COWAIT_WAITALL` - funkcja zwraca S_OK, jeśli wszystkie obiektyw otrzymały sygnał**

Metody `Signal()` i `Reset()` służą do zgłoszenia sygnału do obiektu lub jego skasowania.

Asynchroniczne wywoływanie metod Przykład użycia

```
ICalculator* pCalculator;
ICallFactory* pCallFactory;
AsyncICalculator* pAsyncCalculator;
ISynchronize* pSynchronize;

CoCreateInstance(CLSID_Calculator,NULL,CLSCTX_ALL,
                 IID_ICalculator, (void**)&pCalculator);

pCalculator->QueryInterface(IID_CallFactory, (void**)&pCallFactory);

pCallFactory->CreateCall(IID_AsyncICalculator, NULL,
                         (void**)&pAsyncCalculator);

pAsyncCalculator->QueryInterface(IID_ISynchronize,
                                 (void**)&pSynchronize);

pAsyncCalculator->Begin_Compute(...);

while (pSynchronize->Wait(0,0) == RPC_S_CALLPENDING){
        ... // do some job
}

pAsyncCalculator->End_Compute(...);
```