Programowanie zaawansowanych aplikacji w C#

Procesy. Debugowanie i śledzenie aplikacji.

Jerzy Białkowski

Wydział Matematyki i Informatyki UMK

Wykład 7

Składowe klasy *Process*

Statyczne metody klasy *Process*

- **EnterDebugMode** – przełącza proces w tryb pozwalający na interakcję z processami systemu operacyjnego działającymi w specjalnym trybie (włącza natywne uprawnienia *SeDebugPrivilege* dla bieżącego wątku)
- **GetCurrentProcess** – tworzy nową instancję klasy powiązaną z bieżącym procesem
- **GetProcessById** – tworzy nową instancję klasy powiązaną istniejącym procesem o zadanym identyfikatorze
- **GetProcesses** – tworzy tablicę obiektów powiązanych z procesami działającymi w systemie operacyjnym
- **GetProcessesByName** – tworzy tablicę obiektów powiązanych z procesami o zadanej nazwie działającymi w systemie operacyjnym
- **LeaveDebugMode** – wychodzi ze specjalnego trybu
- **Start** – tworzy nowy proces uruchamiając wskazany program lub otwierając podany dokument i zwraca nowy obiekt klasy *Process* powiązany z tym procesem

Wybrane nie-statyczne metody klasy *Process*

- **Close** – zwalnia zasoby związane z instancją klasy (obiektem)
- **CloseMainWindow** – zamyka proces posiadający interfejs użytkownika poprzez wysłanie polecenia zamknięcia głównego okna aplikacji tego procesu
- **GetService** – zwraca obiekt reprezentujący usługę (jeśli proces realizuje usługę, w przeciwnym wypadku zwraca *null*)
- **Kill** – natychmiast zatrzymuje proces związany z obiektem z którego jest wywoływana metoda
- **OnExited** – wywołuje zdarzenie *Exited*
- **Refresh** – powoduje ponowne odczytanie buforowanych wartości własności obiektu klasy z którego metoda była wywołana
- **WaitForExit** – blokuje bieżący wątek do czasu zakończenia procesu powiązanego z obiektem na którym wywołano metodę lub do upłynięcia czasu przeterminowania podanego w argumencje
- **WaitForInputIdle** – powoduje oczekiwanie na przejście procesu w stan bezczynności
- **Start** – uruchamia proces z programem lub dokumentem wskazywanym przez własność *StartInfo*

Wybrane (nie-statyczne) własności (properties) klasy *Process* (1/5)

- **BasePriority** – pobiera bazowy priorytet dla procesu (wartość liczbową wyliczoną na podstawie wartości własności *PriorityClass*: 4 – Idle, 8 – Normal, 13 – High, 24 – RealTime) (niedostępne dla procesu uruchomionego z własnością *ProcessStartInfo.UseShellExecute* ustawioną na *true*)
- **CanRaiseEvents** – pozwala sprawdzić, czy proces może wywoływać zdarzenia
- **EnableRaisingEvents** – pobiera lub ustawia wartość od której zależy, czy proces w momencie zakończenia procesu powinno być wywoływane zdarzenie *Exited*
- **ExitCode** – pobiera kod zakończenia procesu (po jego zakończeniu)
- **ExitTime** – pobiera czas w którym proces się zakończył
- **Handle** – zwraca natywny uchwyt procesu (przypisany procesowi w momencie startu i używany do utrzymywania jego atrybutów; może być użyty do inicjalizowania uchwytów *WaitHandle* lub wywoływania natywnych metod)
- **HandleCount** – pobiera ilość uchwytów (uchwytów zasobów takich jak uchwyty plików, kolejek komunikatów itp.) otwartych przez proces

Wybrane (nie-statyczne) własności (properties) klasy *Process* (2/5)

- **HasExited** – zwraca wartość pozwalającą na sprawdzenie, czy proces się zakończył
- **Id** – zwraca (unikalny) identyfikator procesu
- **MachineName** – zwraca nazwę maszyny, na której działa proces
- **MainModule** – pobiera główny moduł procesu (moduł użyty do uruchomienia procesu) (niedostępne dla procesu uruchomionego z własnością *ProcessStartInfo.UseShellExecute* ustawioną na *true*)
- **MainWindowHandle** – pobiera uchwyt głównego okna aplikacji procesu (niedostępne dla procesu uruchomionego z własnością *ProcessStartInfo.UseShellExecute* ustawioną na *true*)
- **MainWindowTitle** – pobiera nagłówek głównego okna aplikacji procesu (niedostępne dla j.w.)
- **MaxWorkingSet** – pobiera lub ustawia maksymalną wielkość „working set” (zbioru stron pamięci widocznego dla procesu w fizycznej pamięci RAM)
- **MinWorkingSet** – pobiera lub ustawia minimalną wielkość „working set”
- **Modules** – pobiera tablicę/kolekcję modułów wczytanych przez proces

Wybrane (nie-statyczne) własności (properties) klasy *Process* (3/5)

- **NonpagedSystemMemorySize64** – pobiera rozmiar niestronicowanej pamięci zaalokowanej przez proces
- **PagedMemorySize64** – pobiera rozmiar stronicowanej pamięci zaalokowanej przez proces
- **PagedSystemMemorySize64** – pobiera rozmiar stronicowanej pamięci systemowej zaalokowanej dla procesu
- **PeakPagedMemorySize64** – pobiera maksymalny rozmiar pamięci zaalokowanej w pamięci wirtualnej pliku wymiany (od czasu uruchomienia)
- **PeakVirtualMemorySize64** – pobiera maksymalny rozmiar pamięci wirtualnej używanej przez proces
- **PeakWorkingSet64** – pobiera maksymalny rozmiar pamięci fizycznej używanej przez proces
- **PriorityBoostEnabled** – pobiera lub ustawia wartość indykującą, czy priorytet skojarzonego procesu powinien być tymczasowo zwiększany (przez system operacyjny) w czasie gdy główne okno procesu jest aktywne (jest wybrane)

Wybrane (nie-statyczne) własności (properties) klasy *Process* (4/5)

- **PriorityClass** – pozwala pobierać lub ustawiać klasę priorytetu dla procesu (typ wyliczeniowy *ProcessPriorityClass*: Normal/Idle/High/RealTime/BelowNormal/AboveNormal)
- **PrivateMemorySize64** – pobiera ilość pamięci zaalokowanej przez proces która nie może być współdzielona z innymi procesami
- **PrivilegedProcessorTime** – pobiera ilość czasu jaką proces działał w obrębie rdzenia systemu operacyjnego (czas systemowy)
- **ProcessName** – pobiera nazwę procesu
- **ProcessorAffinity** – pobiera lub ustawia maskę reprezentującą procesory na których mogą działać wątki procesu
- **Responding** – zwraca wartość informującą o tym, czy interejs użytkownika procesu reaguje (na bieżąco!) na wejście użytkownika
- **SessionId** – pobiera numer sesji (terminala/usługi terminalowej) procesu
- **StandardError** – pobiera strumień do odczytu z wyjścia błędów aplikacji

Wybrane (nie-statyczne) własności (properties) klasy *Process* (5/5)

- **StandardInput** – pobiera strumień do zapisu na standardowe wejście aplikacji
- **StandardOutput** – pobiera strumień do odczytu ze standardowego wyjścia aplikacji
- **StartInfo** – pobiera lub ustawia dane z którymi uruchamiany jest proces (np. argumenty z linii wywołania, nazwa pliku (dokumentu) do otwarcia, nazwa użytkownika, hasło, profil użytkownika)
- **StartTime** – pobiera czas uruchomienia procesu
- **Threads** – pobiera lub przypisuje tablicę wątków działających w procesie
- **TotalProcessorTime** – pobiera łączny czas procesora działania procesu
- **UserProcessorTime** – pobiera „czas użytkownia” działania procesu
- **VirtualMemorySize64** – pobiera ilość pamięci wirtualnej zaalokowanej przez proces
- **WorkingSet64** – pobiera ilość fizycznej pamięci zaalokowanej przez proces

Lista konstrukcji w przykładowych programach

Lista konstrukcji do zaprezentowania na wykładzie w przykładowych programach

- wybieranie bieżącego procesu
- wybieranie procesu po identyfikatorze
- wybieranie procesu po nazwie
- listowanie wszystkich procesów z systemu (i ich własności)
- tworzenie nowych procesów
- wypisywanie własności procesu – sprawdzanie zmian własności podczas działania procesu (pokazanie, że niektóre własności są buforowane (cachowane))
- sprawdzanie zmian własności procesu – sprawdzanie zmian własności podczas działania procesu i odczyt wartości maksymalnych (szczytowych)
- listowanie modułów załadowanych w wybranym z uruchomionych procesów (i ich własności)

Składowe klasy *ProcessStartInfo*

konstruktory klasy *ProcessStartInfo*

- **ProcessStartInfo()**
- **ProcessStartInfo(string)**
  (nazwa programu lub dokumentu do uruchomienia)
- **ProcessStartInfo(string, string)**
  (nazwa programu lub dokumentu oraz argumenty)

Wybrane własności (properties) klasy *ProcessStartInfo* (1/4)

- **Arguments** – podaje lub pobiera argumenty z linii wywołania
- **CreateNoWindow** – informuje, czy proces ma być uruchamiany w nowym oknie
- **Domain** – domena (Active Directory) która ma być użyta przez uruchamiany proces

Wybrane własności (properties) klasy *ProcessStartInfo* (2/4)

- **EnvironmentVariables** – pobiera zmienne środowiskowe
- **ErrorDialog** – pobiera lub ustawia wartośc odpowiedzialną za to, czy ma być otwierane okienko dialogowe z informacją o błędzie jeśli proces nie może zostać uruchomiony
- **ErrorDialogParentHandle** – pobiera lub ustawia uchwyt okna otwieranego w przypadku, gdy proces nie może zostać uruchomiony
- **FileName** – pobiera lub ustawia nazwę programu lub dokumentu do „uruchomienia”
- **LoadUserProfile** – pobiera lub ustawia wartość odpowiedzialna za to, czy dane profilu użytkonika mają być wczytywane z rejestrów
- **Password** – pobiera lub ustawia bezpieczny łańcuch zawierający hasło używane podczas uruchmiania procesu

Wybrane własności (properties) klasy *ProcessStartInfo* (3/4)

- **RedirectStandardError** – pobiera lub ustawia wartość odpowiedzialna za to, czy standardowe wyjście błędów ma zostać przekierowane do strumienia *Process.StandardError*
- **RedirectStandardInput** – pobiera lub ustawia wartość odpowiedzialna za to, czy standardowe wyjście uruchamianego procesu ma zostać przekierowane do strumienia *Process.StandardInput*
- **RedirectStandardOutput** – pobiera lub ustawia wartość odpowiedzialna za to, czy standardowe wejście uruchamianego procesu ma zostać przekierowane do strumienia *Process.StandardOutput*
- **StandardErrorEncoding** – pobiera lub ustawia domyślny format kodowania standardowego wyjścia błędów
- **StandardOutputEncoding** – pobiera lub ustawia domyślny format kodowania standardowego wyjścia

Wybrane własności (properties) klasy *ProcessStartInfo* (4/4)

- **UserName** – pobiera lub ustawia nazwę użytkownika używaną podczas uruchmiania procesu
- **UseShellExecute** – pobiera lub ustawia wartość odpowiedzialna za to, czy przy uruchamianiu procesu ma być używana powłoka systemu operacyjnego
- **Verb** – pobiera lub ustawia akcję jaka ma być podejmowana przy uruchamianiu procesu z dokumentem (zadanym przez własność *FileName*)
- **Verbs** – pobiera tablicę akcji jakie mogą być podjęte przy uruchamianiu procesu z dokumentem (zadanym przez własność *FileName* – rozpoznawane po rozszerzeniu)
- **WindowStyle** – pobiera lub ustawia stan okna z jakim ma być uruchamiany proces
- **WorkingDirectory** – pobiera lub ustawia ścieżkę do katalogu roboczego dla uruchamianego procesu

Klasy związane z licznikami i sprawdzaniem stosu

Klasy związane z licznikami

- **PerformanceCounter** – klasa implementująca licznik systemowy
- **PerformanceCounterCategory** – klasa implementująca kategorię liczników
- **CounterCreationData** – klasa opisująca dane potrzebne do utworzenia licznika
- **CounterCreationDataCollection** – klasa implementująca kolekcję danych potrzebną do utworzenia kategorii liczników
- **CounterSample** – struktura reprezentująca próbkę danych licznika

Klasy związane ze sprawdzaniem stanu stosu

- **StackTrace** – klasa pozwalająca na wydobywanie informacji o ramkach stosu
- **StackFrame** – klasa przechowująca informacje o ramce stosu

Składowe klasy *PerformanceCounter*

Wybrane metody klasy *PerformanceCounter*

- **BeginInit** – rozpoczyna inicjalizację instancji licznika;
- **Close** – zamyka licznik i zwalnia przydzielone przez tę instancję licznika zasoby;
- **CloseSharedResources** – zwalnia zasoby z biblioteki współdzielonej zaalokowane przez licznik;
- **Decrement** – w atomicznej (niepodzielnej) operacji zmniejsza wartość licznika o 1;
- **EndInit** – kończy inicjalizację instancji licznika;
- **Increment** – w atomicznej (niepodzielnej) operacji zwiększa wartość licznika o 1;
- **IncrementBy** – w atomicznej (niepodzielnej) operacji zwiększa wartość licznika o zadaną wartość;
- **NextSample** – pobiera próbkę danych i zwraca jej surową lub nieobliczoną wartość;
- **NextValue** – pobiera próbkę danych i zwraca jej wyliczoną wartość;
- **RemoveInstance** – usuwa instancję ***kategorii*** do której przypisany jest licznik (o nazwie wyznaczonej przez własność InstanceName).

Składowe klasy *PerformanceCounter*

Wybrane własności (properties) klasy *PerformanceCounter*

- **CategoryName** – pobiera lub ustawia nazwę kategorii licznika dla licznika;
- **CounterName** – pobiera lub ustawia nazwę licznika skojarzoną z instancją licznika;
- **CounterType** – pobiera typ skojarzonego licznika;
- **InstanceLifetime** – pobiera lub ustawia czas życia procesu;
- **InstanceName** – pobiera lub ustawia nazwę instancji dla licznika;
- **MachineName** – pobiera lub ustawia nazwę maszyny dla licznika;
- **RawValue** – pobiera lub przypisuje surową lub niewyliczoną wartość licznika;
- **ReadOnly** – pobiera lub ustawia wartość odpowiedzialną za to czy licznik jest w trybie wyłącznego odczytu.

Typy liczników

Pola typu wyliczeniowego *PerformanceCounterType*

- **NumberOfItems32**
- **NumberOfItems64**
- **NumberOfItemsHEX32**
- **NumberOfItemsHEX64**
- **RateOfCountsPerSecond32**
- **RateOfCountsPerSecond64**
- **CountPerTimeInterval32**
- **CountPerTimeInterval64**
- **RawFraction**
- **RawBase**
- **AverageTimer32**
- **AverageBase**
- **AverageCount64**
- **SampleFraction**
- **SampleCounter**
- **SampleBase**
- **CounterTimer**
- **CounterTimerInverse**
- **Timer100Ns**
- **Timer100NsInverse**
- **ElapsedTime**
- **CounterMultiTimer**
- **CounterMultiTimerInverse**
- **CounterMultiTimer100Ns**
- **CounterMultiTimer100NsInverse**
- **CounterMultiBase**
- **CounterDelta32**
- **CounterDelta64**

Podział liczników

Podział liczników ze względu na rodzaj wykonywanych obliczeń

Liczniki możemy podzielić ze względu na rodzaj wykonywanych obliczeń na następujące rodzaje

- wyliczające **średnią** wartość pomiarów (ang. *average*); z każdym takim licznikiem jest skojarzony licznika bazowy (ang. base) zliczający ilość pomiarów;
- wyliczające **różnicę** dwóch ostatnich pomiarów (and. *difference*); jeśli jest ona dodatnia to jest ona zwracana jako wynik, w przeciwnym wypadku zwracana jest wartość zerowa;
- zwracające najbardziej **aktualny** pomiar (ang. *instantaneous*);
- zwracające wartość w postaci **procentowej** (ang. *percentage*);
- wyliczające **tempo** zliczeń w jednostce czasu (ang. *rate*).

Klasy przeznaczone dla debugowania aplikacji

Klasy przeznaczone dla debugowania aplikacji

- **Debugger**
- **Debug**

Uwaga:

Klasy *Debug* „działa” tylko dla projektów zbudowanych w trybie *Debug*.
Programy budowane w innych trybach (np. *Release*) będą miały usuwane konstrukcje używające wspomnianej klasy.

Alternatywa debugowania aplikacji – śledzenie

Klasy przeznaczone dla śledzenia aplikacji

- **Tracer**
- **BooleanSwitch**
- **TraceSwitch**
- **TraceListener**
- **TraceListenerCollection**

Klasy służące do obsługi wyjścia debugowania i śledzenia

- **DefaultTraceListener** generuje poprzez metody *Write* i *Writeline* komunikaty kierowane do metod *OutputDebugString* oraz *Debugger.Log*, komunikaty te są wyświetlane w Visual Studio w oknie *Output*, analogicznie przetwaczanie są komunikaty *Fail* i *Assert*; jako jedyny jest automatycznie załączany do każdej kolekcji *Listeners* (pozostałe wymagają dodania do kolekcji *Listeners* w celu użycia)
- **EventLogTraceListener** przekierowuje wyjście do (systemowego) logu zdarzeń
- **ConsoleTraceListener** przekierowuje „śledzone” lub „debugowane” wyjście albo na standardowe wyjście albo na standardowe wyjście błędów

Klasy służące do obsługi wyjścia debugowania i śledzenia

- **TextWriterTraceListener** przekierowuje wyjście do instacji klasy *TextWriter* lub *Stream* (w szczególności pozwala na zapisa na konsolę i do pliku bo są one reprezentowane przez instancje klasy *Stream*)
- **DelimitedListTraceListener** – przekierowuje wyjście do obiektu klasy *TextWriter* lub *Stream* (np. *FileStream*), którego szczegóły mogą zostać podane przez własności *Delimiter*
- **XmlWriterTraceListener** – zapisuje wyjście w postaci danych *XML* do obiektów klasy *TextWriter* lub *Stream*