

Mgr. Jiří Činčura
jiri@x2develop.com
blog.cincura.net

„free lunch“

- * Hrubý výkon CPU již neroste jako dříve
 - * teplo
- * Přidávají se další sofistikované jednotky
 - * Hyperthreading
- * Především celá jádra
 - * Vícejádrové procesory
- * Stroje s více procesory

Aktuální trendy

- * Využití více výpočetních jednotek těžké
 - * Efektivní
- * Mnoho procesů, ale pracuje jich jen málo a neefektivně
- * Potřeba využít celý výpočetní výkon
 - * V budoucnu 80+ jader/CPU

Vlákná (thready)

- * Spolehlivost systému
- * Bezpečnost (izolace)
- * Škálovatelnost
- * Robustnost
- * Virtualizace CPU

Vlákná (thready)

- * Vlákna jsou drahé objekty
 - * Vytvoření, zrušení
 - * 200 000 cyklů vytvoření, 100 000 zrušení
- * Thread kernel object
 - * Vlastnosti vlákna
- * Thread environment block (TEB, 4KB)
 - * Exception-handling chain, TLS, GDI/OpenGL
- * User-mode stack
 - * 1MB, CLR committed
- * Kernel-mode stack (12KB/24KB)
- * DLL attach/detach notifikace

Vlákná (thready)

- * V systému mnoho vláken
 - * Nejlepší vlákna = počet CPU
 - * Většina vláken nic nedělá
- * Context switching
 - * 6000-8000 cyklů
- * CLR vlákna mapována na Windows vlákna

Vlákna (thready)

- * Windows scheduler
 - * Preemptive multitasking
 - * Běží vlákna, ne procesy
- * Time slice/quantum
- * Switching
- * Priority
 - * 0 až 31
- * Stárnutí
- * Priority boost
- * Priorita procesu (6)
- * Priorita vlákna (7)
- * <http://msdn.microsoft.com/en-us/library/ms685100.aspx>

CPU bound operace

- * Task
 - * Fine grained paralelismus
 - * Interně využíván ThreadPool
 - * Mnohem „kulturnější“ API
- * Result
- * Wait
- * ContinueWith

CPU bound operace

- * Zrušení
 - * CancellationToken svázan s taskem
 - * Nemusí být ani spuštěna (ale naschedulována)
- * Child tasky
 - * Hierarchie operací
 - * Parent čeká na child

CPU bound operace

- * Task
 - * WaitAll
 - * WaitAny
- * TaskFactory
 - * StartNew
 - * ContinueWhenAll
 - * ContinueWhenAny
 - * FromAsync

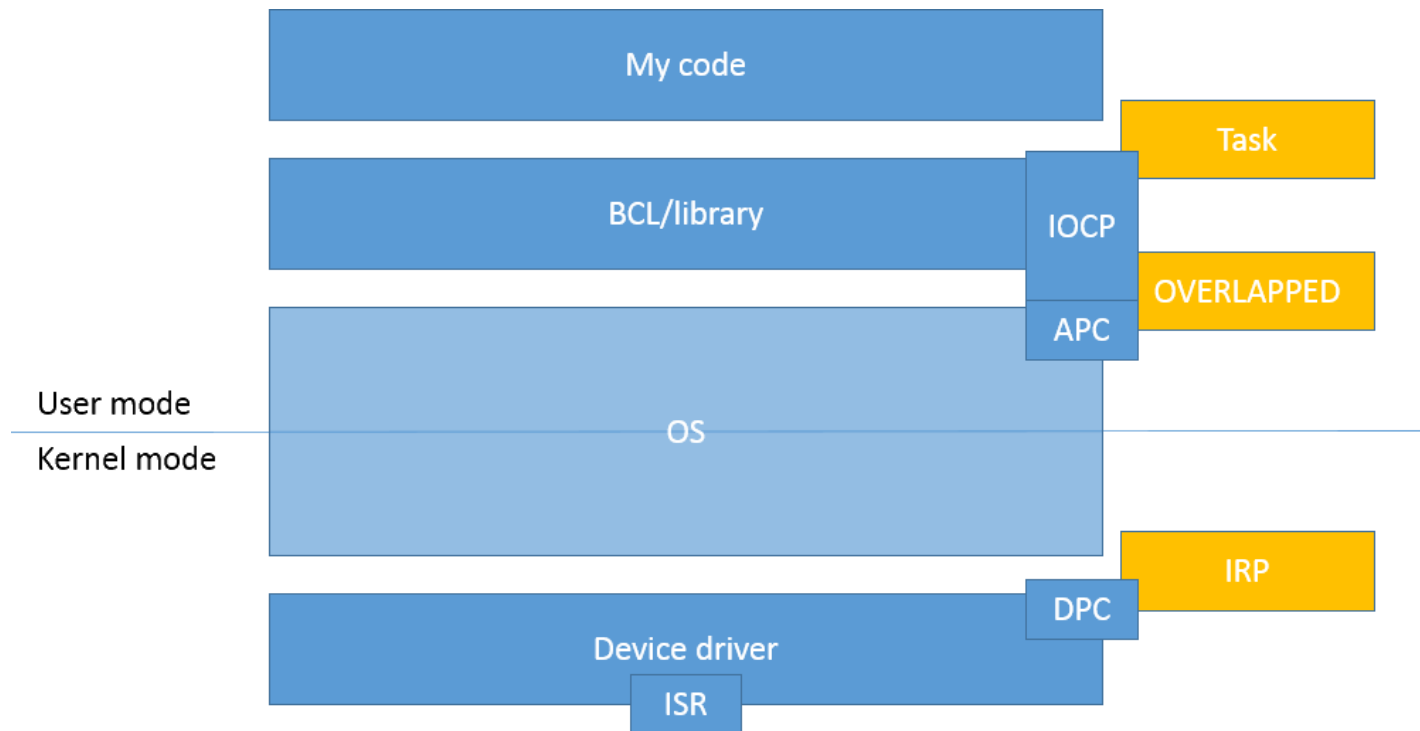
I/O bound operace

- * I/O operaci provádí HW ne CPU
- * Systém je informován o výsledku
- * Thread je blokován a nic nedělá
 - * Plýtvání zdroji
- * Asynchronní operace jsou efektivnější

Async/Await

- * Dvě nová klíčová slova v .NET 4.5
- * Async pouze označuje asynchronní metodu, „nic“ nedělá
- * Asynchronní ➔ neblokující
 - * Žádná vlákna
 - * Latence

Async/Await



Async/Await

- * Zjednodušení práce s BeginXxx/EndXxx
 - * Exceptions, smyčky, ...
- * Kompilátor zajišťuje správné poskládání metod
 - * Callbacky
 - * Stavový automat (podobné jako IEnumerable)

Zamykání, koordinace

- * Nejzajímavější část vícevláknového/paralelního programování
- * Ovlivňuje výslednou efektivitu řešení
- * Izolace více vláken od konkurenční změny sdílených dat a jejich poškození
- * Zamykat co nejméně
- * Na co nejkratší + rozumnou dobu
- * Zamykání je drahá operace

Zamykání, koordinace

- * User-mode
 - * Rychlejší než kernel-mode
 - * Speciální CPU instrukce
 - * Systém neví o blokování (plýtvání CPU)
 - * Vhodné pro krátké operace
- * Kernel-mode
 - * Poskytuje jádro systému
 - * Přejíždí user-mode → kernel-mode
 - * Systém blokuje vlákno
 - * Při dostupnosti zdroje je vlákno systémem probuzeno
 - * Cross process
- * Hybridní

Zamykání, koordinace

```
* internal static class StrangeBehavior
* {
*     private static Int32 s_stopWorker = 0;
*     public static void Main()
*     {
*         Console.WriteLine("Main: letting worker run for 5 seconds");
*         Thread t = new Thread(Worker);
*         t.Start();
*         Thread.Sleep(5000);
*         s_stopWorker = 1;
*         Console.WriteLine("Main: waiting for worker to stop");
*         t.Join();
*     }
*     private static void Worker(Object o)
*     {
*         Int32 x = 0;
*         while (s_stopWorker == 0)
*             x++;
*         Console.WriteLine("Worker: stopped when x={0}", x);
*     }
* }
```

Zamykání, koordinace

```
* class OutOfProgramOrder
* {
*     private Int32 m_flag = 0;
*     private Int32 m_value = 0;
*
*     public void Thread1()
*     {
*         // These could execute in reverse order
*         m_value = 5;
*         m_flag = 1;
*     }
*
*     public void Thread2()
*     {
*         // m_value could be read before m_flag
*         if (m_flag == 1)
*             Display(m_value); // o!
*     }
* }
```

Q & A

