

برنامه‌نویسی وظیفه‌ای در
.NET 4.0 & 4.5

با استفاده از

TPL & PLINQ

مهندس سید منصور عمرانی
انتشارات پندار پارس

سرشناسه	: عمرانی، سید منصور، ۱۳۵۶ -
عنوان و نام پدیدآور	: برنامه‌نویسی وظیفه‌ای در NET 4.0 و NET 4.5. با استفاده از TPL و PLINQ: چگونه برنامه‌هایی سریع‌تر و کاربرپسندتر بنویسیم/ ترجمه و تالیف سید منصور عمرانی.
مشخصات نشر	: تهران: پندار پارس، ۱۳۹۱.
مشخصات ظاهری	: ۶۹۶ ص: مصور، جدول.
شابک	: 978-600-6529-12-7: ۱۹۸۰۰۰ ریال
موضوع	: مایکروسافت ویژوال سی شارپ دات نت
موضوع	: سی شارپ (زبان برنامه‌نویسی کامپیوتر)
موضوع	: برنامه‌نویسی شی‌گرا
موضوع	: برنامه‌نویسی موازی
رده بندی کنگره	: QA۷۶۷۳/س۱۳۹۱۹۵ ۸۳ع
رده بندی دیویی	: ۱۳۳/۰۰۵
شماره کتابشناسی ملی	: ۲۷۳۵۰۱۷

انتشارات پندار پارس



دفتر فروش: انقلاب، ابتدای کارگر جنوبی، کوی رشتچی، شماره ۱۴، واحد ۱۶ www.pendarepars.com
 تلفن: ۶۶۵۷۲۳۳۵ - تلفکس: ۶۶۹۲۶۵۷۸ همراه: ۰۹۱۲۲۴۵۲۳۴۸
info@pendarepars.com



نام کتاب	: برنامه‌نویسی وظیفه‌ای در NET 4.0 & 4.5. با استفاده از TPL & PLINQ
ناشر	: انتشارات پندار پارس ناشر همکار: پارشمن
ترجمه و تالیف	: سید منصور عمرانی
چاپ نخست	: بهار ۹۱
شمارگان	: ۱۰۰۰ نسخه
طرح جلد	: رامین شکرالهی
لیتوگرافی، چاپ، صحافی	: ترام سنچ، صالحان، خیام

قیمت : ۱۹۸۰۰ تومان به همراه CD شابک : ۹۷۸-۶۰۰-۶۵۲۹-۱۲-۷



* هرگونه کپی برداری، تکثیر و چاپ کاغذی یا الکترونیکی از این کتاب بدون اجازه ناشر تخلف بوده و پیگرد قانونی دارد *

مقدمه

دنیای برنامه‌نویسی و نرم‌افزار همیشه دستخوش تغییر و تحول‌های جدید بوده است. از معرفی کلاس‌ها و مولفه‌های جدید گرفته تا معرفی فناوری‌های جدید نرم‌افزاری و شیوه‌ها و الگوهای جدید برنامه‌نویسی. در این میان یکی از تحولات تاثیرگذار بر دنیای برنامه‌نویسی در سال‌های اخیر تغییر و تحول پردازنده‌ها و سکوه‌های سخت‌افزاری بوده است. به طوری که دیگر عصر تولید پردازنده‌های تک هسته‌ای به سر رسیده و عصر پردازنده‌های چند هسته‌ای و چندپردازنده‌ها آغاز شده است.

این تغییر به نوبه‌ی خود شرکت‌های رهبری کننده‌ی فناوری‌های نرم‌افزار را به تکاپو انداخته است. زیرا اگر کسی علاقه دارد برنامه‌اش در سکوه‌های جدید سریع‌تر کار کند، باید دیدگاه برنامه‌نویسی خود را تغییر بدهد و موازی فکر کند. علاوه بر این پاسخ‌پذیر بودن برنامه‌ها در ادواتی مانند Tablet PC یا Touch بسیار مهم است و راه حل آن نیز فقط اجرای موازی است. در واقع تا جای ممکن باید کارهای زمان‌بر را که باعث قفل شدن موقت واسط کاربر یا یخ زدن فرم‌ها می‌شود به صورت موازی اجرا کنیم.

در این میان شرکت مایکروسافت با معرفی کتابخانه‌ی اختصاصی برنامه‌نویسی موازی به نام TPL در NET 4.0. گوی سبقت را از دیگر شرکت‌ها ربوده است. خصوصاً اضافه شدن کلمات کلیدی `await` و `async` به زبان‌های NET. و مجموعه‌ی وسیعی از متدهای غیر همزمان به کتابخانه‌ی کلاس‌ها در NET 4.5. نشان می‌دهد مایکروسافت توجه و عظمی جدی به این مقوله دارد. با این اوصاف می‌توان گفت آشنایی با TPL به نوعی اجتناب‌ناپذیر است و کسانی که می‌خواهند در NET 4.5. برنامه‌نویسی کنند حتماً باید با TPL و الگوی TAP آشنا باشند.

محتوای کتاب

کتاب «برنامه‌نویسی وظیفه‌ای در NET 4.0 & 4.5». با استفاده از TPL و PLINQ» حاصل 8 ماه تلاش و زحمت مستمر در خصوص شیوه‌ی جدید برنامه‌نویسی موازی در NET 4.0 و NET 4.5. است. محتوای کتاب ترکیبی از ترجمه، تالیف و تحقیق است. مولف در نوشتن کتاب از 12 کتاب لاتین استفاده نموده و با تحقیق و مطالعه‌ی مقالات بیشماری در اینترنت و مستندات MSDN، مجموعه‌ای را گردآوری کرده که بدون اغراق می‌توان گفت از نظر جامعیت مشابه ندارد. منابع اصلی مولف در تالیف این کتاب به شرح زیر بوده است:

#	عنوان	ناشر	مؤلف	سال انتشار
1	Essential C# 4.0	Addison-Wesley	Mark Michaelis	2010
2	Pro .NET 4.0 Parallel Programming in C#	Apress	Adam Freeman	2010
3	Accelerated C# 2010	Apress	Trey Nash	2010
4	Pro C# 2010 and .NET 4.0 Platform	Apress	Andrew Troelsen	2010
5	C# 4.0 The Complete Reference	McGraw-Hill	Herbert Schildt	2010
6	CLR via C# 3 rd Edition	MicrosoftPress	Jeffrey Richter	2010
7	C# 4.0 in a Nutshell	OReilly	Josep Albahari Ben Albahari	2010
8	Practical .NET2 and C# 2	ParadoxalPress	Patrick Smacchia	2006
9	Professional Parallel Programming with C#	Wrox	Gaston C.Hillar	2011

#	عنوان	ناشر	مؤلف	سال انتشار
10	Windows Internals 5 th Edition	Microsoft Press	Mark Russinovich David Solomon	2009
11	Essential COM	Addison-Wesley	Don Box	1997
12	Patterns of Parallel Programming in .NET (C#)	Stepehn Toub	Microsoft	2010
13	Task-Based Asynchronous Pattern	Stephen Toub	Microsoft	2010

مخاطبین

مخاطبین این کتاب تمام کسانی هستند که به طور کلی تحت سکوی NET. برنامه‌نویسی می‌کنند، زیرا برنامه‌نویسی موازی منحصر به نوع به خصوصی از برنامه‌ها نیست. در این کتاب شیوه‌ی قدیمی و سنتی و همچنین شیوه‌ی مدرن برنامه‌نویسی موازی در NET 4.0 و NET 4.5. با استفاده از مثال‌های عملی متعدد به صورت اصولی و زیر بنایی آموزش داده می‌شود. این کتاب خصوصاً برای برنامه‌نویسان برنامه‌های تحت ویندوز و همچنین افرادی که قصد افزایش کارایی برنامه‌های تحت سرویس‌دهنده‌ی خود را دارند بسیار مفید است. علاوه بر این کتاب مزبور می‌تواند برای دانشجویان رشته‌ی مهندسی نرم‌افزار در زمینه‌ی درس سیستم عامل، برنامه‌نویسی موازی و تا حدی کامپایلر مفید باشد.

نیازمندی‌ها

خواننده برای مطالعه و فراگیری مباحث این کتاب به داشتن تجربه یا تخصصی ویژه و طولانی در کار با NET. نیازی ندارد. اما باید با اصول کلی برنامه‌نویسی و شیء‌گرایی آشنا باشد. زبان مقصد این کتاب C# است. لذا خواننده باید در حد متوسط با این زبان آشنا باشد و بتواند یک برنامه‌ی NET. را در Visual Studio اجرا کند. مثال‌های این کتاب عمدتاً برنامه‌های ساده‌ی Console Application هستند، اما گهگاه از برنامه‌های Windows Forms نیز استفاده شده است. با توجه به این مساله خواننده باید آشنایی نسبی و مختصری با نوشتن یک برنامه‌ی تحت ویندوز توسط Visual Studio را داشته باشد.

همچنین بحث این کتاب به نسخه‌ی خاصی از NET. منحصر نمی‌شود و تمامی نسخه‌های NET. را در بر می‌گیرد. اما سعی شده هر جا کلاس یا قابلیت به نسخه‌ی به خصوصی از NET. نیاز داشته باشد، شماره‌ی نسخه‌ی NET. ذکر شود. در صورتی که خواننده از Visual Studio 2010 استفاده کند می‌تواند تمام مثال‌های کتاب را (به جز مثال‌های فصل 34 و 35) بدون هیچ مشکلی اجرا کند. مابقی کدها (فصل 34 و 35) به قابلیت‌های خاص NET 4.5. بر می‌گردد که با توجه به این که هنوز این نسخه از NET. به طور رسمی توسط مایکروسافت ارائه نشده نمی‌توان آنها را با استفاده از Visual Studio 2010 اجرا کرد. اما خواننده می‌تواند با دانلود نسخه‌ی آزمایشی Visual Studio 11 آنها را آزمایش کند و از قبل خود را مهیای قابلیت‌های برنامه‌نویسی موازی NET 4.5. نماید.

ساختار کتاب

این کتاب در قالب 6 بخش و 40 فصل تالیف شده و هر فصل به بحث مجزایی می‌پردازد. شیوه‌ی ارائه‌ی مطلب در این کتاب، آموزش عملی همراه با مثال است. لذا هر فصل با مثال‌های زیادی همراه است اما با توجه به تعدد موضوعات مطرح شده، این کتاب را به عنوان مرجع نیز می‌توان استفاده کرد. فصل‌های کتاب بر اساس ترتیب مشخصی از لحاظ یادگیری و آموزش چیده شده به طوری که هر فصل نوعاً پیش نیاز فصل بعدی محسوب می‌شود.

به همین دلیل روش مورد انتظار مطالعه‌ی کتاب، خواندن آن از ابتدا تا انتها بر اساس شماره‌ی فصل‌ها است. با این وجود اگر خواننده با برخی مباحث آشنا باشد و بخواهد یک فصل به خصوص را مطالعه کند می‌تواند مستقیماً به آن مراجعه کند و الزامی وجود ندارد حتماً فصل‌های قبل از آن را هم بخواند (اگرچه توصیه می‌شود این کار را بکند). اما در هر فصل فرض می‌شود خواننده قبلاً با مباحث پیش‌نیاز آن آشنایی داشته باشد.

موضوع فصل‌ها نیز از یکدیگر کاملاً مجزا است و از ارائه‌ی چندین موضوع در یک فصل خودداری شده است. به همین دلیل است که تعداد فصول کتاب زیاد است. اما مزیت این شیوه‌ی تقسیم‌بندی این است که خواننده به سرعت می‌تواند به موضوع خاصی در کتاب مراجعه کرده و آن را مطالعه نماید. تلاش بسیاری صورت گرفته تا محتوای هر فصل کاملاً جامع باشد و تمام نکات مرتبط با موضوع را پوشش بدهد. به طوری که بعد از خواندن آن چیز دیگری باقی نماند که خواننده باید بداند. سعی شده فصل‌ها تا جای ممکن کوتاه باشد، با این حال در برخی از فصول نظیر فصول 29 تا 32 که به الگوهای برنامه‌نویسی غیر همزمان مربوط می‌شود، برای ادا شدن حق مطلب محتوا کمی مفصل است.

سی‌دی ضمیمه

این کتاب مشتمل بر حدود 200 مثال عملی است که تمام آنها در قالب پروژه‌های مجزا و قابل اجرا در سی‌دی ضمیمه قرار داده شده است. علاوه بر این می‌توانید مثال‌های کتاب را در صفحه‌ی اختصاصی آن از سایت انتشارات پندار پارس دانلود نمایید.

بازخورد

پرداختن به مقوله‌ی برنامه‌نویسی چندنخی و موازی کار بسیار سختی است و نوشتن یک کتاب جامع در این باره بسیار سخت‌تر است. مولف با خواندن کتاب‌های مختلف و مقالات و مستندات بیشمار در رابطه با هر موضوع، حداکثر تلاش خود را به کار برده تا به طور کاملاً جامع به تمامی مباحث برنامه‌نویسی موازی در سکوی NET. بپردازد. لذا از طرح ابهام‌آمیز یا ناقص مطلب خودداری نموده است. قطعاً این کتاب خالی از اشکال نیست ضمن این که موضوعات دیگری نیز وجود دارد که در این کتاب قابل طرح است. حتماً نظرات، انتقادات، سوالات و پیشنهادات خود را در رابطه با این کتاب با مولف از طریق سایت انتشارات (www.pendarepars.com) یا آدرس mansoor.omrani@yahoo.com در میان بگذارید. مولف در انتها امیدوار است این کتاب برای جامعه‌ی برنامه‌نویسان مفید بوده و از خواندن آن لذت ببرید.

سید منصور عمرانی

بهار 1391

تقدیم بہ پدر و مادری مہربان
کہ محبتی بی دریغ نثارم کردند

فهرست کتاب در یک نگاه

صفحه	عنوان	فصل
3	تاریخچه‌ی پردازنده‌ها و برنامه‌نویسی هم‌روند	1
19	مفاهیم اولیه	2
39	پردازه‌ها و دامنه‌ی برنامه‌ها	3
53	برنامه‌نویسی چندنخی سنتی: قسمت اول	4
69	برنامه‌نویسی چندنخی سنتی: قسمت دوم	5
89	حوضچه‌ی نخ	6
99	مبانی هماهنگ‌سازی	7
113	دستور lock	8
121	هماهنگ‌سازی با استفاده از Monitor	9
135	سیگنال‌دهی با Monitor	10
143	دستگیره‌ی انتظار	11
155	دستگیره‌ی انتظار رویداد	12
165	موانع حافظه و فرآری	13
175	عملیات اتمی پایه	14
185	ساختارهای دو رگه	15
197	قفل‌های خواندن و نوشتن	16
205	راه‌اندازی کُند به شکل ایمن	17
213	انبار محلی نخ یا TLS	18
233	تایمرها	19
243	فراخوانی غیر همزمان نماینده‌ها یا ADI	20
257	لغو کردن عملیات هم‌روند و الگوی لغو مشارکتی	21
273	توسعه‌های موازی یا PFX	22
279	توازی و وظیفه‌ای	23
303	وظایف تو در تو و متوالی	24
325	سایر مباحث مرتبط با برنامه‌نویسی وظیفه‌ای	25
345	کلاس Parallel	26
۳۶۵	Parallel LINQ	27
389	قسمت‌بندی	28
423	کلکسیون‌های هم‌روند	29
447	الگوی برنامه‌نویسی غیر همزمان APM: قسمت اول	30
461	الگوی برنامه‌نویسی غیر همزمان APM: قسمت دوم	31
485	الگوی غیر همزمان مبتنی بر رویداد یا EAP: قسمت اول	32
513	الگوی غیر همزمان مبتنی بر رویداد یا EAP: قسمت دوم	33
537	الگوی غیر همزمان وظیفه‌ای یا TAP: قسمت اول	34
561	الگوی غیر همزمان وظیفه‌ای یا TAP: قسمت دوم	35
581	نخ‌های .NET و فناوری COM	36
599	برنامه‌نویسی موازی و واسط کاربر	37
619	آزمایش، بررسی کارایی و اشکال‌زدایی برنامه‌های هم‌روند	38
643	دستورات SIMD و سایر کتابخانه‌های برنامه‌نویسی موازی	39
657	اختتامیه و جمع‌بندی	40

فهرست

3	فصل 1. تاریخچه‌ی پردازنده‌ها و برنامه‌نویسی هم‌روند
3	غول بی‌شاخ و دم برنامه‌نویسی چندنخی
3	آغاز سفر
4	چند پردازنده‌ها و پردازنده‌های چند هسته‌ای
4	سیر تولید پردازنده‌ها
5	انواع فناوری پردازنده‌ها
6	وضعیت فعلی و آینده
6	معماری سیستم‌ها
6	تک پردازنده‌های چند هسته‌ای
7	معماری پردازنده‌های چند هسته‌ای
8	چند پردازنده‌ها و معماری NUMA
9	ویندوز و تعداد پردازنده‌ها
11	فرمول محاسبه‌ی تعداد پردازنده‌ی منطقی
11	معماری NUMA و نرم‌افزار
12	سیستم‌های توزیع شده
13	بهره‌وری از پردازنده
14	برنامه‌نویسی چندنخی و موازی در .NET و C#
14	روش جدید در .NET 4.0
15	تفاوت بنیانی روش سنتی و روش جدید
16	چرا روش سنتی نیز در این کتاب توضیح داده شده است؟
16	یک مثال عملی
19	فصل 2. مفاهیم اولیه
19	مروری بر مفاهیم پایه‌ای .NET
19	پردازه
19	دامنه
20	نخ
20	چندنخی
21	ارتباط AppDomain و نخ
21	ارتباط AppDomain و اسمبلی
21	نخ‌های .NET و ویندوز
22	نخ سخت‌افزاری و نخ نرم‌افزاری
23	انواع مختلف هم‌روندی و اجرای نخ‌های نرم‌افزاری
24	برش زمانی و تعویض متن
24	وابستگی پردازنده‌ای
25	ساختار نخ
25	شیء کرنلی نخ
27	بلوک محیطی نخ یا TEB
28	موقع ایجاد نخ چه اتفاقی می‌افتد؟
28	آگاه‌سازی DLL ها
29	زمان بند
30	زمان بند چطور کار می‌کند؟
30	موقع تعویض متن چه اتفاقی می‌افتد؟
31	خنثی کردن اثر کش پردازنده
32	با تعویض متن چه بکنیم؟
32	مدت زمان تعویض متن چقدر است؟

32	نگاهی به وضعیت پردازها و نخها در ویندوز.....
33	تبعات ناشی از استفاده‌ی بی‌رویه از نخها
34	مشکلات برنامه‌های چند نخی
37	نتیجه‌گیری
37	موارد استفاده‌ی هم‌روندی.....
39	فصل 3. پردازها و دامنه‌ی برنامه‌ها
39	پردازها در NET.....
39	کلاس Process.....
43	کلاس ProcessThread.....
45	کلاس ProcessStartInfo.....
47	دامنه‌ی برنامه
47	مروری بر اعضای مهم کلاس AppDomain.....
49	کاربرد عملی: بارگذاری اسمبلی به همراه نمونه‌سازی اشیاء در یک AppDomain دیگر.....
49	ارتباط Remoting با نخها
52	اداره‌ی استثناء‌های اداره نشده.....
53	فصل 4. برنامه‌نویسی چند نخی سستی قسمت اول
53	مقدمه
53	کلاس Thread.....
55	انواع نخ
55	خصوصیت Thread.IsBackground.....
56	مشخصات نخ.....
57	متد استاتیک Thread.Sleep() و خواباندن موقت نخ
57	انصراف آنی از برهه‌ی زمانی
58	متد استاتیک Thread.Yield().....
58	حوزه‌ی دید اشیاء نخ.....
59	دسترسی به شیء نخ جاری.....
60	نقطه‌ی خاتمه‌ی برنامه و لزوم انتظار برای خاتمه‌ی نخ
62	انتظار برای خاتمه‌ی اجرای یک نخ با استفاده از متد Join().....
64	مبادله‌ی اطلاعات با نخ.....
65	ارسال آرگومان به نخ.....
66	دریافت اطلاعات از نخ.....
67	نسخه‌های مختلف Join() و پرهیز از انتظار نامحدود
69	فصل 5. برنامه‌نویسی چند نخی سستی قسمت دوم
69	مقدمه
69	اولویت نخها.....
69	نحوه‌ی زمان‌بندی نخها
69	انواع حالات مختلف اولویت نخها
70	اولویت پردازه
71	اولویت Realtime.....
71	اولویت‌های بالا.....
71	اولویت‌های پنهان.....
72	اداره‌ی استثناء در نخها.....
72	استثناء‌های اداره نشده در AppDomain.....
73	استثناء‌های اداره نشده در برنامه‌های Windows Forms.....
75	وقفه و بیدار کردن اجباری

76	ایجاد وقفه برای نخ‌های در حال کار
78	متوقف کردن کامل نخ
80	اطلاع از Abort() شدن نخ
81	ارسال آرگومان به نخ هنگام Abort()
82	امتناع از Abort()
84	مقایسه Interrupt() و Abort()
84	تعلیق و ادامه دادن نخ
85	خصوصیت Thread.ThreadState و وضعیت‌های مختلف نخ
87	حداکثر تعداد نخ‌های یک پردازنده
88	خلاصه
89	فصل 6. حوضچه‌ی نخ
89	مقدمه
89	حوضچه‌ی نخ
89	نحوه‌ی استفاده از حوضچه‌ی نخ برای اجرای هم‌روند
90	حوضچه‌ی نخ چگونه کار می‌کند؟
91	انواع نخ‌های حوضچه‌ای
91	نحوه‌ی کار مدیر حوضچه
91	الگوریتم حوضچه‌بندی و نقطه‌ی اشباع
91	بهبود حوضچه‌ی نخ‌ها در NET 4.0
92	دستکاری نخ‌های حوضچه‌ای
92	انتظار برای خاتمه‌ی کار نخ‌های حوضچه‌ای
93	محدودیت‌های نخ‌های حوضچه‌ای
94	چه چیزهایی در NET از نخ‌های حوضچه‌ای استفاده می‌کنند؟
95	ظرفیت حوضچه
96	چه موقع از نخ‌های سنتی استفاده کنیم؟
99	فصل 7. مبانی هماهنگ‌سازی
99	مقدمه
99	هماهنگ‌سازی
100	امنیت نخ
102	تامین امنیت نخ
103	رقابت و ناحیه‌ی بحرانی
105	نحوه‌ی هماهنگ‌سازی نخ‌ها در رابطه با ناحیه‌ی بحرانی
106	بن‌بست
108	کاربرد دیگری از هماهنگ‌سازی
109	چه موقع به هماهنگ‌سازی نیاز داریم؟
110	انتخاب ساختار هماهنگ‌سازی درست
110	انواع ساختارهای هماهنگ‌سازی از نظر ساختار درونی
111	انواع ساختارهای هماهنگ‌سازی از نظر عملکرد
112	انواع بن‌بست از نظر ساختارهای هماهنگ‌سازی
112	خلاصه
113	فصل 8. دستور lock
113	دستور lock
115	متغیر قفل و ملزومات آن
117	رابطه‌ی ناحیه‌ی بحرانی، متغیر قفل و منبع مشترک
118	محافظت از چندین ناحیه‌ی بحرانی توسط یک شیء قفل

119	اتمی بودن عملیات محافظت شده توسط دستور lock
120	قفل‌های تو در تو یا بازگشتی
120	خلاصه
121	فصل 9. هماهنگ‌سازی با استفاده از Monitor
121	لاس Monitor
121	نحوه‌ی استفاده از Monitor برای حفاظت از ناحیه‌ی بحرانی
123	فراخوانی بازگشتی
123	آزاد کردن قطعی قفل
124	شیوه‌ی درست استفاده از کلاس Monitor و نحوه‌ی پیاده‌سازی دستور lock
125	متد Monitor.TryEnter()
126	چرا شیء قفل باید ارجاعی باشد؟
127	طرز کار کلاس Monitor
129	چند نمونه از کاربرد Monitor توسط CLR
129	سازنده‌ی استاتیک یا سازنده‌ی نوع
130	ویژگی [MethodImplAttribute]
131	چرا در دستور lock یا کلاس Monitor نباید از this یا Type ها به عنوان شیء قفل استفاده کنیم؟
132	مشکل کلاس Monitor در استفاده از Type به عنوان قفل
132	مشکل کلاس Monitor در استفاده از this به عنوان قفل
133	مشکلات Monitor و lock و نکات ایمنی در کار با آنها
134	کارایی Monitor
135	فصل 10. سیگنال‌دهی با Monitor
135	مقدمه
135	انتظار و سیگنال‌دهی با Wait() و Pulse()
137	مشکل خواب ابدی در Wait/Pulse
138	نحوه‌ی استفاده‌ی صحیح از Wait() و Pulse()
139	مشکل همچنان ادامه دارد
141	انتظار محدود
143	فصل 11. دستگیره‌ی انتظار
143	مقدمه
143	مشکلات قابلیت سیگنال‌دهی کلاس Monitor
144	کلاس WaitHandle
144	نحوه‌ی استفاده از WaitHandle و ساختارهای کرنلی
145	متدهای استاتیک WaitAny(), WaitAll() و SignalAndWait()
146	شیء دوجانبه‌ی انحصاری یا Mutex
147	نحوه‌ی ایجاد Mutex
148	نحوه‌ی استفاده از Mutex
149	Mutex و قابلیت بازگشت
149	کاربرد Mutex برای جلوگیری از اجرای مکرر برنامه
151	سمافور
151	سمافور چگونه کار می‌کند؟
152	نحوه‌ی استفاده از سمافور
155	فصل 12. دستگیره‌ی انتظار رویداد
155	مقدمه
155	کلاس EventWaitHandle
155	کلاس AutoResetEvent
155	نحوه‌ی استفاده

156	باز نگه داشتن در بدون این که کسی پشت در ایستاده باشد
158	پیاده‌سازی دروازه‌ی خودکار بازگشتی
158	کلاس ManualResetEvent
160	انتظار محدود
161	از بین بردن در
161	ایجاد دستگیره‌های انتظار مشترک در بین چندین پردازنده
161	دستگیره‌ی انتظار و صرفه‌جویی در مصرف منابع
162	کلاس‌های قابل استفاده توسط ThreadPool.RegisterWaitForSignalObject()
163	نسخه‌های مختلف ThreadPool.RegisterWaitForSignalObject()
164	خاتمه یا لغو انتظار
164	خلاصه
165	فصل 13. موانع حافظه و فرآری
165	مقدمه
165	حصارهای حافظه و فرآری
166	کامپایلر و بهینه‌سازی
167	اثرات جانبی بهینه‌سازی
168	تغییر ترتیب اجرای دستورات و رفتار پردازنده
169	جلوگیری از رفتار غیر منتظره‌ی برنامه‌ها
170	نحوه‌ی استفاده از ساختارهای فرآر و حصارهای حافظه
172	موارد استفاده‌ی حصار کامل توسط کامپایلر C#
172	متغیرهای فرآر و کلمه‌ی کلیدی volatile
173	مشکلات متغیرهای فرآر
175	فصل 14. عملیات اتمی پایه
175	کلاس Interlocked
176	نوشتن متدهای ریاضی اتمی بر اساس Interlocked
178	نوشتن یک ساختار هماهنگ‌سازی شخصی بر اساس Interlocked
179	مشکل ساختار SimpleSpinLock
179	ساختار SpinWait
180	ساختار SpinLock
181	نحوه‌ی استفاده از SpinLock
181	خصوصیت‌های ساختار SpinLock
182	سایر مشخصات SpinLock
183	فرق SpinLock با lock
183	کلاس‌های دارای رویداد
185	فصل 15. ساختارهای دو رگه
185	مقدمه
185	نحوه‌ی طراحی
185	قابلیت‌ها و مزایا
186	ساختارهای دو رگه
186	کلاس ManualResetEventSlim
188	کلاس SemaphoreSlim
189	کلاس CountdownEvent
189	طرز کار CountdownEvent
191	کلاس Barrier
192	نحوه‌ی استفاده از Barrier
193	سایر قابلیت‌های Barrier

196	خلاصه‌ی مشخصات و قابلیت‌های ساختارهای دو رگه
197	فصل 16. قفل‌های خواندن و نوشتن
197	مقدمه
197	طرز کار قفل‌های خواندن و نوشتن
198	کارایی قفل‌های خواندن و نوشتن
198	نحوه‌ی استفاده از ReaderWriterLockSlim
200	قفل‌های ارتقاء‌پذیر
202	فراخوانی تو در تو یا بازگشت
202	مشکلات ReaderWriterLock
202	جمع‌بندی
205	فصل 17. راه‌اندازی کند به شکل ایمن
205	مقدمه
206	بهبود دادن تکنیک قفل‌گذاری با دوبار بررسی
208	کلاس‌های راه‌اندازی کند ایمن در NET
208	کلاس <T>Lazy
210	کلاس LazyInitializer
213	فصل 18. انبار محلی نخ یا TLS
213	نحوه‌ی استفاده از انبار محلی نخ
214	ویژگی [ThreadStaticAttribute]
214	مشکل مقدار اولیه‌ی فیلدهای استاتیک
216	کلاس <T>ThreadLocal
217	<T>ThreadLocal و فیلدهای نمونه‌ای
218	انبار محلی نخ و نخ‌های حوضچه‌ای
220	متدهای GetData() و SetData() در کلاس Thread
223	کلاس CallContext
224	مقایسه و جمع‌بندی
224	نخ منطقی
227	زمینه‌ی اجرا یا ExecutionContext
228	بهبود برنامه به وسیله‌ی جلوگیری از به جریان افتادن زمینه‌ی اجرا
229	متن امنیتی نخ و جعل هویت
231	تسخیر متن
233	فصل 19. تایمرها
233	مقدمه
233	انواع تایمر
234	کلاس System.Threading.Timer
235	نحوه‌ی کار System.Threading.Timer و عوارض آن
235	رفع مشکل فراخوانی تداخلی callback
237	کلاس System.Timers.Timer
239	دقت تایمرهای چندنخی و تایمر چند رسانه‌ای ویندوز
241	تایمرهای تک نخ
243	فصل 20. فراخوانی غیر همزمان نماینده‌ها یا ADI
243	مشکلات اجرای هم‌روند متدها در روش برنامه‌نویسی چندنخی سنتی
243	نماینده‌ها
244	فراخوانی غیر همزمان نماینده‌ها
245	جزئیات ماجرا

245	نماینده‌های غیر همزمان و استثناء
246	درک بهتر نحوه‌ی کار نماینده‌های غیر همزمان با پیاده‌سازی یک کلاس شخصی
248	امضاء متدهای BeginInvoke() و EndInvoke()
249	کسب اطلاع از خاتمه‌ی عملیات
249	1. انتظار به شکل دستی
251	2. روش پاس دادن توالی یا CPS
251	مشکلات و حواشی روش پاس دادن توالی
254	استفاده از نماینده‌های ژنریک <>Func و <>Action
256	خاتمه‌ی بحث
257	فصل 21. لغو کردن عملیات هم‌روند و الگوی لغو مشارکتی
257	الگوی لغو مشارکتی
257	مدل لغو مشارکتی مایکروسافت و کلیات آن
258	استفاده از مدل لغو مشارکتی در برنامه‌نویسی چندنخی سنتی
258	مرحله‌ی 1: لغو عملیات هم‌روند با چک کردن توکن لغو
259	مرحله‌ی 2: آگاه شدن از لغو عملیات هم‌روند با تولید استثناء
260	مرحله‌ی 3: منعکس کردن استثناء به نخی که منتظر خاتمه‌ی عملیات هم‌روند است
261	تعریف یک کلاس عمومی برای عملیات هم‌روندی که قابل لغو شدن باشد
262	جمع‌بندی نحوه‌ی پیاده‌سازی مدل لغو مشارکتی
263	موارد استفاده از مدل لغو مشارکتی در NET
266	توکن لغو چگونه کار می‌کند؟
267	لغو زنجیره‌ای
268	توکن لغوی با چندین منبع
269	خصوصیت WaitHandle و متد Register()
270	ترکیب کردن لغو سنتی نخ‌ها با الگوی لغو مشارکتی
273	فصل 22. توسعه‌های موازی یا PFX
273	چرا PFX؟
273	دو رهیافت در برنامه‌نویسی موازی
274	توازی ساختنیافته و غیر ساختنیافته
274	PFX چیست؟
274	TPL
275	PLINQ
275	اجزاء PFX
276	قابلیت‌های جدید برنامه‌نویسی موازی در NET 4.0
277	چرا TPL و PLINQ ارزشمند هستند؟
279	فصل 23. توازی وظیفه‌ای
279	مقدمه
279	کلاس Task
279	ایجاد و اجرای وظیفه
281	نکات کلی در رابطه با شیء وظیفه
281	وضعیت وظیفه
283	کارخانه‌ی وظایف TaskFactory
285	انتظار برای خاتمه‌ی اجرای یک وظیفه
286	تنظیمات وظیفه
287	منهدم کردن وظیفه
287	وظیفه‌ای با مقدار برگشتی
288	اداره‌ی استثناء توسط وظایف

290	Task و استثناء‌های کاملاً هضم شده
291	انتظار برای اجرای چندین وظیفه
293	لغو کردن اجرای وظیفه
293	مرحله 1: تغییرات اولیه‌ی مورد نیاز در برنامه‌ی اصلی
294	مرحله 2: تغییرات مورد نیاز داخل عملیات موازی
295	مرحله 3: تغییرات نهایی مورد نیاز در برنامه‌ی اصلی
297	آرگومان token و نحوه‌ی دسترسی به آن در عملیات موازی
298	راه حل اول
299	راه حل دوم
299	پس پارامتر CancellationToken در سازنده‌ی Task به چه دردی می‌خورد؟
300	راه حل‌های دیگر
300	راه حل سوم
301	راه حل چهارم
301	مروری بر کلاس Task
302	خلاصه
303	فصل 24. وظایف تو در تو و متوالی
303	مقدمه
303	وظایف تو در تو
304	وظایف پدر و فرزند
305	مزیت تعریف رابطه‌ی پدر و فرزند
305	مقایسه‌ی وظایف تو در تو و وظایف پدر و فرزند
306	لغو کردن زنجیره‌ای
308	وظیفه‌ی متوالی
309	عدم لزوم تعریف صریح وظیفه‌ی قبلی
309	پارامتر Task به چه دردی می‌خورد؟
310	زنجیره‌ی فراخوانی وظایف متوالی
311	وظیفه‌ی متوالی و پردازش ترتیبی
312	تنظیمات وظیفه‌ی متوالی
315	سایر نسخه‌های متد ContinueWith()
316	وظیفه‌ای با چندین وظیفه‌ی متوالی
317	وظیفه‌ی متوالی وظایف بسیار سریع
317	وظیفه‌ی متوالی و وضعیت وظیفه‌ی قبلی
318	استثناء در وظایف متوالی
318	توالی مشروط
320	توالی پس از وظایف متوالی مشروط
321	وظیفه‌ی متوالی با چندین وظیفه‌ی قبلی
322	وظایف متوالی و ژنریک‌ها
323	لغو کردن اجرای وظیفه‌ی متوالی
323	مروری بر وظایف متوالی
324	خلاصه
325	فصل 25. سایر مباحث مرتبط با برنامه‌نویسی وظیفه‌ای
325	مقدمه
325	کلاس زمان‌بند یا TaskScheduler
325	انواع زمان‌بند
325	زمان‌بند پیش فرض
326	نگاهی دقیق‌تر به حوضچه‌ی نخ

326	صف سراسری و صف محلی
327	دزدیدن کار
328	نگاهی دیگر به برخی جزئیات نسبتاً غامض در خصوص وظیفه‌ها
329	زمان‌بند هماهنگ‌ساز متن
329	زمان‌بندهای سفارشی
330	نحوه‌ی استفاده از زمان‌بند
331	آخرین سنگر به دام اندازی استثناء‌های اداره نشده‌ی وظایف
333	جلوگیری از درهم شکسته شدن برنامه
334	کارخانه‌ی وظایف یا TaskFactory
336	کلاس <TaskCompletionSource<TResult>
339	متد Task.Unwrap()
340	بررسی دقیق‌تر استثنای AggregateException
340	متدهای Handle() و Falltern()
342	داخل Task چه خبر است؟
342	پهپود TPL در NET 4.5
343	خلاصه
345	فصل 26. کلاس Parallel
345	مقدمه
345	متد Parallel.Invoke()
346	Parallel.Invoke() و ایمنی نخی
347	تنظیمات Parallel.Invoke()
348	Parallel.Invoke() و استثناء
348	متد Parallel.For()
349	شکستن حلقه‌ی Parallel.For()
350	متوقف کردن آنی حلقه‌ی Parallel.For()
350	متد Parallel.ForEach()
352	پیاپی‌سازی حلقه‌ی که گام (Step) شمارنده‌اش بیش از 1 است
353	حلقه‌های تو در تو
353	حلقه‌ی ForEach() اندیس‌دار
354	نگاه دقیق‌تری به شکستن حلقه با استفاده از ParallelLoopState
355	تشخیص شکسته شدن حلقه بعد از اتمام آن
355	رفتار عجیب Break()
356	فرق Break() و Stop()
356	بی‌اعتمادی در توقف و شکستن حلقه
358	پهپود حلقه‌ها در رابطه با مقادیر محلی
360	اداره‌ی استثناء
360	لغو اجرای حلقه‌ی موازی از بیرون
361	تفاوت لغو عملیات از بیرون در کلاس Parallel و Task
362	چه موقع باید از Parallel.For() و Parallel.ForEach() استفاده کرد؟
364	تعیین حداکثر درجه‌ی اجرای موازی
365	فصل 27. Parallel LINQ
365	مقدمه
365	اصول کلی PLINQ
365	نحو پرس و جوها
365	منبع داده و حوزه‌ی عملکرد
366	نحوه‌ی کار

367	مقایسه‌ی LINQ و PLINQ.....
369	متدهای توسعه‌ی کلیدی در PLINQ.....
370	متد AsParallel().....
371	متد Range().....
372	متد Repeat().....
372	متد AsOrdered().....
374	تفاوت AsOrdered() با OrderBy().....
375	متد AsUnordered().....
375	PLINQ و نحوه‌ی اجرای پرس و جو.....
375	اجبار کردن اجرای موازی توسط WithExecutionMode().....
376	محدود کردن درجه‌ی هم‌روندی با استفاده از WithDegreeOfParallelism().....
376	فراخوانی توابع پس‌زمینه یا درگیر در I/O.....
377	اجبار کردن اجرای ترتیبی با استفاده از AsSequential().....
378	متد WithMergeOptions().....
379	متد ForAll().....
380	پرس و جوهای PLINQ و امنیت نخی.....
381	PLINQ و اجرای دیر هنگام.....
381	متدهای ToDictionary(), ToList(), ToArray() و.....
382	وقوع استثناء در یک پرس و جوی PLINQ.....
383	لغو اجرای پرس و جوی PLINQ.....
384	پهپود عملیات انجمنی شخصی.....
387	محدودیت‌های PLINQ.....
387	چه موقع از PLINQ استفاده کنیم؟.....
389	فصل 28. قسمت‌بندی.....
389	قسمت‌بندی و لزوم آن.....
390	درک عملی فرآیند قسمت‌بندی.....
391	عوامل مهم در ابداع یک الگوریتم قسمت‌بندی.....
391	تعداد آیتم‌ها.....
392	نحوه‌ی دسترسی به آیتم‌ها.....
392	مشکل شمارشگرها در پردازش موازی.....
394	تایمن امنیت نخی شمارشگرها.....
395	بار کاری آیتم‌ها و کلکسیون.....
395	نحوه‌ی بخش بار در طول کلکسیون.....
397	الگوریتم‌های مختلف قسمت‌بندی.....
397	قسمت‌بندی بازه‌ای.....
399	قسمت‌بندی هاشورزنی.....
400	قسمت‌بندی تکه‌ای.....
400	نحوه‌ی پیاده‌سازی.....
402	انواع الگوریتم‌های قسمت‌بندی تکه‌ای.....
403	مقایسه‌ی الگوریتم قسمت‌بندی بازه‌ای و تکه‌ای.....
403	الگوریتم‌های هوشمند.....
404	قسمت‌بند.....
406	الگوریتم‌های استاتیک و داینامیک.....
406	مبهم بودن مفهوم استاتیک و داینامیک.....
407	قسمت‌بندی در PLINQ و کلاس Parallel.....
407	استراتژی‌های مختلف قسمت‌بندی در PLINQ.....
409	استراتژی‌های مختلف قسمت‌بندی در کلاس Parallel.....

410	قسمت‌بندها در TPL
411	کلاس‌های قسمت‌بند <TSource> Partitioner و <TSource> OrderablePartitioner
411	شخصی‌سازی قسمت‌بندی
415	نوشتن یک قسمت‌بند شخصی
419	ایندکس‌های نا امن
420	تحمیل استراتژی قسمت‌بندی به PLINQ
421	ترکیب PLINQ و کلاس Parallel
422	خلاصه
423	فصل 29. کلکسیون‌های هم‌روند
423	مقدمه
423	کلکسیون‌های هم‌روند
424	واسط <T> IProducerConsumerCollection
425	سایر متدهای کلکسیون‌های هم‌روند
427	<T> ConcurrentBag
428	کلاس <TKey, TValue> ConcurrentDictionary
429	امنیت نخ‌های متدهای دیکشنری هم‌روند
429	<T> BlockingCollection
430	نحوه‌ی استفاده از <T> BlockingCollection
430	تولید آیت‌م جدید
430	مصرف آیت‌م‌ها
431	خاتمه‌ی تولید و مصرف
433	مروری بر کلاس <T> BlockingCollection
434	کار با چندین <T> BlockingCollection
439	چه زمانی از کلکسیون‌های هم‌روند استفاده کنیم؟
440	صف هم‌روند/صف معمولی
440	پشته‌ی هم‌روند/پشته‌ی معمولی
440	دیکشنری هم‌روند/دیکشنری معمولی
441	کیف هم‌روند
441	کلکسیون بلاک‌کننده
441	هم‌روندی در کلکسیون‌های غیر هم‌روند
441	هم‌روندی در کلکسیون‌های غیر ژنریک
443	هم‌روندی در کلکسیون‌های ژنریک
447	فصل 30. الگوی برنامه‌نویسی غیر هم‌زمان قسمت اول
447	مقدمه
447	درک مفهوم برنامه‌نویسی غیر هم‌زمان و بهره‌وری در منابع
449	اصول الگوی برنامه‌نویسی غیر هم‌زمان
449	امضای متدهای غیر هم‌زمان
450	کلاس‌هایی که در NET. مدل APM را پیاده‌سازی کرده‌اند
450	نحوه‌ی استفاده از متدهای APM
450	کسب اطلاع از خاتمه‌ی عملیات غیر هم‌زمان
451	1. انتظار به شکل دستی
451	2. روش پاس دادن توالی یا CPS
453	مشکلات و حواشی روش پاس دادن توالی
454	اداره‌ی استثناء در مدل APM
455	تکنیک‌های میعادگاه
455	لغو اجرای متد غیر هم‌زمان
456	نگاهی به واسط IAsyncResult

457	مقایسه‌ی متدهای غیر همزمان و فراخوانی غیر همزمان نماینده‌ها
457	متدهای غیر همزمان و برنامه‌نویسی وظیفه‌ای
458	مزایای FromAsync()
458	خلاصه‌ی مزایای TaskFactory.FromAsync():
459	کلاس FileStream و عملیات غیر همزمان
461	فصل 31. الگوی برنامه‌نویسی غیر همزمان قسمت دوم: کاربردها
461	مقدمه
461	نوشتن یک برنامه‌ی سرویس‌دهنده/سرویس‌گیرنده‌ی ارتباط لوله‌ای
461	نسخه‌ی همزمان (غیر APM)
463	نسخه‌ی غیر همزمان (APM)
465	صفحات وب غیر همزمان
470	نحوه‌ی پیاده‌سازی الگوی APM در یک کلاس شخصی
471	شیوه‌ی نادرست پیاده‌سازی متدهای APM
472	روش صحیح پیاده‌سازی متدهای APM
472	پیاده‌سازی یک الگوریتم غیر همزمان به روش APM و مشکلات آن
472	شیء IAsyncResult
475	استفاده از IAsyncResult یک شیء APM دیگر
476	چگونه شیء APM را بین BeginXXX و EndXXX به اشتراک بگذاریم؟
477	متدی با چندین شیء APM
482	مشکل Tuple ها
483	اداره‌ی خطا
483	مروری بر اصول کلی پیاده‌سازی الگوی APM در یک کلاس
484	APM و واقعیت‌ها
485	فصل 32. الگوی غیر همزمان مبتنی بر رویداد یا EAP قسمت اول: معرفی
485	مقدمه
486	معرفی EAP
487	سخن کلی EAP چیست؟
487	اهداف EAP
487	اصول کلی الگوی EAP
488	مثالی از یک کلاس EAP: کلاس PictureBox
491	ساختار یک کلاس در الگوی EAP
492	نحوه‌ی استفاده از یک کلاس EAP
493	مثال دیگری از یک کلاس EAP: کلاس WebClient
494	فراخوانی تداخلی و کاربرد شیء وضعیت
495	کلاس‌های EAP در NET
495	مولفه‌ی BackgroundWorker
496	نحوه‌ی کار
496	نحوه‌ی استفاده
500	ارسال آرگومان به عملیات غیر همزمان
500	برگرداندن مقدار
501	لغو عملیات
503	قابلیت گزارش پیشرفت
506	جلوگیری از بمباران شدن نخ واسط کاربر توسط رویداد ProgressChanged
507	سیل عظیم رویداد ProgressChanged در برنامه‌های غیر GUI
508	حواشی و مشکلات الگوی EAP
513	فصل 33. الگوی غیر همزمان مبتنی بر رویداد یا EAP قسمت دوم: پیاده‌سازی
513	مقدمه

513	نکاتی که هنگام پیاده‌سازی الگوی EAP باید رعایت کنیم
513	الف. کلاس XXXCompletedEventArgs
515	ب. نماینده‌های XXXCompletedEventHandler و XXXProgressChangedEventHandler
516	ج. نحوه‌ی انجام عملیات غیر همزمان
517	د. فراخوانی تداخلی (XXXAsync)
518	ه. کارهایی که موقع خاتمه‌ی عملیات غیر همزمان باید انجام شود
518	و. گزارش پیشرفت و کلاس XXXProgressChangedEventArgs
520	ز. لغو عملیات
521	استفاده از کلاس‌های AsyncOperation و AsyncOperationManager برای پیاده‌سازی الگوی EAP
522	الگوی EPT: الگوی کاملی برای نوشتن کلاس‌های EAP
527	یک مثال عملی
533	مشکلات الگوی EPT
534	EAP و واقعیت‌ها
534	مقایسه‌ی EAP و APM: چه موقع از مدل EAP استفاده کنیم
537	فصل 34. الگوی غیر همزمان وظیفه‌ای یا TAP و NET 4.5. قسمت اول
537	مقدمه
537	الگوی برنامه‌نویسی غیر همزمان وظیفه‌ای
537	نام‌گذاری، پارامترها و مقدار برگشتی
538	مقدار برگشتی
539	عملیات غیر همزمان و استثناء
539	نحوه‌ی انجام عملیات
541	انواع متدهای TAP
542	لغو عملیات
544	نوشتن نسخه‌ی TAP متد WebClient.DownloadStringAsync()
545	نحوه‌ی استفاده از متدهای TAP
546	متدهای TAP و استثناء مشاهده نشده
546	گزارش پیشرفت
547	مزیت مدل گزارش پیشرفت در الگوی TAP نسبت به الگوی EAP
548	کلاس <T>Progress
549	روش جدید برنامه‌نویسی غیر همزمان در NET 4.5
549	C# 5.0 و کلمات کلیدی await و async
551	نوشتن متدهای TAP دست دوم بر اساس await و async
551	چند نکته‌ی مهم درباره‌ی await و async
553	await و وقوع استثناء در متدهای TAP
554	متدهای async و خطای استفاده
554	await و لغو اجرای متدهای TAP
555	ملاحظات استفاده از await
559	گزارش پیشرفت متدهای TAP
561	فصل 35. الگوی غیر همزمان وظیفه‌ای یا TAP و NET 4.5. قسمت دوم
561	مقدمه
561	ترکیب‌گرها
561	Task.Run()
562	Task.FromResult()
562	Task.WhenAll()
563	Task.WhenAny()
567	Task.Delay()
569	نوشتن ترکیب‌گرهای شخصی
569	متد RetryOnFault()
570	متد NeedOnlyOnce()

570.....	متد (WhenAllOrFirstException)
570.....	ساختار داده‌های وظیفه‌ای.....
570.....	کلاس <AsyncCach<TKey, TValue>.....
572.....	TAP و سایر الگوهای برنامه‌نویسی غیر همزمان.....
572.....	تبدیل APM به TAP.....
573.....	تبدیل TAP به APM.....
573.....	تبدیل TAP به EAP.....
574.....	نوشتن یک متد (WaitOne) بر اساس الگوی TAP برای WaitHandle.....
574.....	استفاده از Task به جای WaitHandle.....
575.....	یک مثال عملی.....
581.....	فصل 36. نخ‌های .NET و فناوری COM
581.....	مقدمه.....
581.....	اساساً COM چیست؟.....
582.....	چند اصل کلی درباره‌ی COM.....
582.....	فناوری COM و Win32.....
583.....	مدل نخ‌ی Win32.....
583.....	مدل نخ‌ی در COM.....
584.....	واقعاً آپارتمان چیست؟.....
584.....	رابطه‌ی نخ‌ها و آپارتمان‌ها.....
584.....	دسترسی به اشیاء COM.....
585.....	انواع مدل چندنخی در COM.....
586.....	دسترسی یک نخ به اشیاء آپارتمان‌های دیگر.....
586.....	نحوه‌ی دسترسی نخ‌های خارجی به اشیاء STA.....
586.....	حلقه‌ی پیام در آپارتمان‌های STA.....
588.....	نحوه‌ی دسترسی نخ STA به اشیاء سایر آپارتمان‌های STA.....
588.....	نحوه‌ی کار آپارتمان‌های MTA.....
589.....	نحوه‌ی کار آپارتمان‌های NTA.....
590.....	مارشالینگ و اجرای راه دور.....
591.....	تفاوت آپارتمان‌ها، نخ‌ها و اشیاء STA و MTA.....
592.....	موقع ایجاد نخ‌های COM چه رخ می‌دهد.....
593.....	مارشالینگ چطور انجام می‌شود؟.....
594.....	اشیاء COM چگونه در ویندوز تعریف می‌شوند؟.....
594.....	قوانین نوشتن مولفه‌های COM.....
595.....	موقع نمونه‌سازی از اشیاء COM چه رخ می‌دهد؟.....
596.....	مدل چندنخی در .NET.....
596.....	.NET و نخ‌ها.....
597.....	منابع.....
599.....	فصل 37. برنامه‌نویسی موازی و واسط کاربر
599.....	مقدمه.....
599.....	مدل نخ‌ی برنامه.....
599.....	وابستگی نخ‌ی.....
599.....	نخ‌های واسط کاربر.....
600.....	روش‌های مختلف دسترسی به المان‌های واسط کاربر از یک نخ دیگر.....
601.....	استفاده از (Invoke) یا (BeginInvoke).....
601.....	مشخص کردن آرگومان‌ها.....
602.....	دریافت مقدار برگشتی.....

602	وقوع استثناء
602	یک مثال عملی
604	استفاده از SynchronizationContext
604	واسط ISynchronizable
604	چگونگی شکل‌گیری SynchronizationContext
605	مفهوم SynchronizationContext
608	جمع‌بندی
609	نحوه‌ی استفاده از SynchronizationContext برای دسترسی به واسط کاربر توسط سایر نرها
611	بر طرف کردن مشکل دسترسی به واسط کاربر در callback های روش APM و ADI برای همیشه
613	استفاده از AsyncOperation
613	استفاده از کلاس‌های EAP نظیر BackgroundWorker
614	استفاده از برنامه‌نویسی وظیفه‌ای و زمان‌بند هماهنگ‌ساز متن
616	دسترسی به واسط کاربر در برنامه‌های موازی NET 4.5
619	فصل 38. آزمایش، بررسی کارایی و اشکال زدایی برنامه‌های هم‌روند
619	مقدمه
619	اهمیت آزمایش
619	اهداف آزمایش
620	اهداف اختصاصی آزمایش برنامه‌های موازی
620	عوامل تاثیرگذار بر برنامه‌های موازی
623	مستندسازی و دستاوردهای آزمایش
624	ابزارهای اشکال‌زدایی، تحلیل و بررسی برنامه‌های موازی
624	Visual Studio Profiling Tools
625	ملزومات ابزار Concurrency Visualization
626	نحوه‌ی تنظیم Microsoft Symbol Server در Visual Studio 2010
626	نحوه‌ی استفاده از ابزار Concurrency Visualization
629	گزارش CPU Utilization
630	گزارش Threads
633	گزارش Cores
633	جمع‌بندی
634	سایر امکانات اشکال‌زدایی برنامه‌های موازی
634	پنجره‌ی Threads
636	پنجره‌ی Parallel Tasks
637	پارامتر state و رفع مشکل بی‌نامی وظیفه‌ها
637	پنجره‌ی Parallel Stacks
638	جعبه ابزار تحلیل کارایی ویندوز یا WPT
639	نحوه‌ی دانلود WPT
640	نحوه‌ی استفاده از WPT
641	WPT و آینده
642	استفاده از چندین مانیتور در Visual Studio 2010
642	منابع فصل
643	فصل 39. دستورات SIMD و سایر کتابخانه‌های برنامه‌نویسی موازی
643	مقدمه
643	دستورات SIMD
644	نسخه‌های مختلف دستورات SIMD
644	کتابخانه‌های اختصاصی استفاده از دستورات SIMD
645	انواع دستورات SIMD: از MMX تا SSE4 و AVX
646	کتابخانه‌ی MKL

647	نحوه‌ی استفاده از کتابخانه‌ی MKL در .NET
651	کتابخانه‌ی IPP
652	نحوه‌ی استفاده از کتابخانه‌ی IPP
655	کتابخانه‌ی ACML
657	فصل 40. اختتامیه
657	مقدمه
657	مروری بر فضای نام System.Threading
657	نوع داده‌های فضای نام System.Threading
660	نوع داده‌های فضای نام System.Threading.Tasks
661	مزایای Task در مقایسه با نخ‌های سنتی
662	مروری بر ساختارهای هماهنگ‌سازی
662	نقاط قوت و ضعف ساختارهای کرنلی
662	نقاط قوت و ضعف ساختارهای مُد کاربری
662	نقاط قوت و ضعف ساختارهای دورگه
664	توصیه‌های کلی در نوشتن برنامه‌های چندنخی و موازی
664	برنامه‌نویسی هم‌روند
666	هماهنگ‌سازی
666	برنامه‌های دارای واسط کاربر
667	چگونه از هماهنگ‌سازی پرهیز کنیم؟
669	جمع‌بندی مدل‌ها و الگوهای برنامه‌نویسی غیر همزمان
670	لینک‌های مفید

بخش 1

برنامه‌نویسی چندنخی سنتی

فصل 1. تاریخچه‌ی پردازنده‌ها و برنامه‌نویسی هم‌روند

فصل 2. مفاهیم اولیه

فصل 3. پردازنده‌ها و دامنه‌ی برنامه‌ها

فصل 4. برنامه‌نویسی چندنخی سنتی: قسمت اول

فصل 5. برنامه‌نویسی چندنخی سنتی: قسمت دوم

فصل 6. حوضچه‌ی نخ

فصل 1. تاریخچه‌ی پردازنده‌ها و برنامه‌نویسی همروند

غول بی‌شاخ و دم برنامه‌نویسی چندنخی

وقت اسم برنامه‌نویسی چند نخی به میان می‌آید بسیاری از برنامه‌نویسان وحشت می‌کنند، زیرا نام این شیوه‌ی برنامه‌نویسی همیشه با یک جور اضطراب و استرس همراه است. مهم نیست چه احساسی نسبت به این مساله دارید، اما واقعیت این است که برنامه‌نویسی همروند مثل راه رفتن در یک میدان پر از مین است! بدون این که علتش را بفهمید ممکن است برنامه‌ای که چندین بار آن را آزمایش کرده بودید و درست کار می‌کرد باگ پیدا کند یا جور دیگری کار کند. باگ‌های برنامه‌های همروند هم یکی از بدخیم‌ترین و پیچیده‌ترین نوع باگ‌ها هستند و تشخیص یا رفع آنها کار بسیار مشکلی است.

اوضاع هم زمانی بدتر می‌شود که تعداد پردازنده‌ها و هسته‌ها بیشتر شود. زیرا برخی از باگ‌های بدذات و موذی تا زمانی که برنامه را روی یک سیستم چند هسته‌ای یا چند پردازنده‌ای اجرا نکنید خودشان را نشان نمی‌دهند. تری نش در کتاب Accelerated C# 2010 داستانی را از تجربه‌ی خودش ذکر می‌کند که قرار بوده نسخه‌ی نهایی یک برنامه را برای شرکت توزیع کننده بفرستند تا آن را بر روی دیسک در تیراژ وسیع تکثیر کند و برای فروش به بازار بفرستد. درست در آخرین لحظات قبل از ارسال برنامه برای شرکت توزیع کننده یکی از برنامه‌نویسان برنامه را روی یک کامپیوتر چند پردازنده اجرا می‌کند و همان لحظه سر و کله‌ی یک باگ پیدا می‌شود.

او می‌گوید آن ماجرا درس بسیار بزرگی برای او و سایر برنامه‌نویسان تیم‌شان بوده است. با این اوصاف شما یا آدم با دل و جراتی هستید و یا خبر ندارید وارد چه سرزمین پر خطری شده‌اید. اما به شما بگویم اگر قدم در این جاده گذاشته‌اید سعی کنید تا آخرش بروید. نصفه و نیمه کار کردن و نوشتن برنامه‌های چند نخی با دانش کم بدترین کاری است که می‌توانید با خودتان بکنید.

آغاز سفر

فکر می‌کنم به عنوان یک پیش در آمد یا مقدمه واقعاً شما را ترسانده باشم. واقعیتش این است که برنامه‌نویسی همروند خیلی وحشتناک نیست، فقط مشکلات و دردسرهای زیادی دارد. اما می‌توانم به شما اطمینان بدهم اگر اصول این کار را بدانید و از خطرات و مشکلات این شیوه‌ی برنامه‌نویسی آگاه باشید، نه تنها ترسناک نیست، بلکه بسیار هم جذاب است. درست مثل بندبازی یا مسابقات اتوموبیل‌رانی سرعت می‌ماند. خطر دارد اما لذت سرعت چیز دیگری است. با استفاده از برنامه‌نویسی موازی هم برنامه‌ی شما واقعاً سریع‌تر می‌شود و هم کاربرپسندی برنامه به میزان بسیار زیادی افزایش پیدا می‌کند. لذا مطمئن باشید از نتیجه‌ی کار حسابی لذت خواهید برد. ابتدا بگذارید نگاهی به سیر تولید پردازنده‌ها و انواع آنها بیندازیم تا ببینیم در چه شرایطی قرار داریم. اطلاع از شرایط موجود اولین کاری است که باید بکنیم.

چند پردازنده‌ها و پردازنده‌های چند هسته‌ای

تا چند سال پیش کامپیوترهای تک پردازنده نظیر پردازنده‌های پنتیوم و آتلون خیلی رواج داشتند. هر بار شرکت‌های اینتل و AMD پردازنده‌ی سریع‌تری بیرون می‌دادند و کاربران به شور و شغف می‌آمدند. می‌توانستید کامپیوتر پنتیوم 3 قدیمی خود را به پنتیوم 4 ارتقاء بدهید و از افزایش سرعت آن واقعاً لذت ببرید. اما چند سالی است که ماجرا تغییر کرده است.

سیر تولید پردازنده‌ها

در سال‌های قبل کورس رقابت بر سر کلاک پردازنده بود و شرکت‌های سازنده‌ی پردازنده مرتباً پردازنده‌های سریع‌تری به بازار می‌دادند. اما کم‌کم متوجه شدند باید سیاست دیگری اتخاذ کنند. چون افزایش سرعت کلاک حرارت زیادی تولید می‌کرد. اگرچه فن‌های قوی‌تر می‌توانستند پردازنده را خنک‌تر کنند، اما فن‌ها نیز بزرگ و بزرگ‌تر می‌شدند. شاید این مساله برای کامپیوترهای رومیزی قابل تحمل باشد اما در کامپیوترهای کیفی چنین چیزی اصلاً شدنی نیست، ضمن این که فن هم باطری مصرف می‌کند. لذا سیاست طراحی پردازنده‌ها تغییر کرد: « اضافه کردن هسته‌های بیشتر ». اما این خبر چندان خوبی برای برنامه‌نویسان معمولی که برنامه‌ها را به شکل عادی (ترتیبی) می‌نوشتند نبود. زیرا در پردازنده‌های چند هسته‌ای برنامه‌ها به شرطی سریع‌تر اجرا می‌شوند که از تمام هسته‌های پردازنده استفاده کنند. به عبارت دیگر به شکل موازی نوشته شوند. چندنخی کردن و موازی نوشتن برنامه هم همان چیزی است که همه احساس بدی نسبت به آن دارند و هر بار به نوعی از زیر بارش در می‌رفتند. اما متأسفانه این بار دیگر نمی‌توانید در بروید. باید مستقیماً با آن روبرو بشوید.

هرب ساتر رئیس استانداردهای ISO زبان ++C که سالها در مورد این زبان و برنامه‌نویسی هم‌روند تجربه و تخصص دارد در سال 2005 در مجله‌ی Dr. Dobbs مقاله‌ای تحت عنوان « The Free Lunch Is Over: A Fundamental Turn Toward Concurrency » نوشت و جامعه‌ی برنامه‌نویسان را به تغییر دیدگاه برنامه‌نویسی برای به دست آوردن حداکثر بهره‌وری از سیستم‌های چند پردازنده‌ی جدید و لزوم این تغییر دیدگاه فراخواند. او توضیح داد برنامه‌نویسان دیگر نمی‌توانند به بهبودی که به صورت مجانی در سرعت پردازنده‌ها توسط شرکت‌های سازنده ایجاد می‌شود اتکا کنند. زیرا تولیدکنندگان پردازنده به جای افزایش فرکانس کلاک، سیاست دیگری در پیش گرفته‌اند (افزایش تعداد هسته‌ها و پردازنده‌ها). مقاله‌ی وی را در آدرس www.gotw.ca/publications/concurrency-ddj.htm می‌توانید مطالعه کنید¹. معنی تمام این حرف‌ها این است که دیدگاه برنامه‌نویسی معمولی باید تغییر کند و از این به بعد باید موازی فکر کنیم. این همان نقطه‌ای است که اکنون در آن قرار داریم.

¹ نقل از کتاب Professional Parallel Programming with C# 2010 از انتشارات Wrox صفحه‌ی 2.

انواع فناوری پردازنده‌ها

به طور کلی سه نوع فناوری وجود دارد:

- **پردازنده‌های فرانخی (Hyperthreading Processors)**

در این فناوری که به اینتل تعلق دارد پردازنده چند سری رجیستر دارد. لذا می‌تواند خودش را به شکل دو (یا چند) پردازنده نشان بدهد. البته در آن واحد فقط یک سری از رجیسترها را می‌تواند استفاده کند، اما می‌تواند طوری خود را به سیستم عامل و نرم‌افزار نشان بدهد که گویی چندین پردازنده‌ی مجزا است. طبق ادعای اینتل این نوع پردازنده‌ها کارایی را حدود 10 تا 30 درصد بهبود می‌دهند.

- **پردازنده‌های چند هسته‌ای (Multi-Core Processors)**

این نوع پردازنده‌ها چند سالی است که وارد صحنه شده‌اند و در حال حاضر پردازنده‌های دو هسته‌ای، چهار هسته‌ای و گهگاه هشت هسته‌ای در بازار مرسوم هستند. اما اینتل دارد روی پردازنده‌های با تعداد هسته‌های بسیار بیشتر مثلاً 80 هسته کار می‌کند. چنین پردازنده‌ای واقعاً قدرت پردازشی بسیار زیادی خواهد داشت. به این نوع پردازنده‌ها، پردازنده‌های چندین هسته‌ای گفته می‌شود.

- **پردازنده‌های چندین هسته‌ای (Manycore Processors)**

مشابه پردازنده‌های چند هسته‌ای هستند و در آنها نیز پردازنده از چندین هسته تشکیل می‌شود اما فرق بین پردازنده‌ی چند هسته‌ای و چندین هسته‌ای به تعداد و نوع هسته‌های آنها بر می‌گردد. در پردازنده‌ی چند هسته‌ای تعداد هسته‌ها حداکثر 8 تا است و تمام هسته‌ها مشابه هم بوده و همگن (homogenous) هستند اما تعداد هسته‌های پردازنده‌ی چندین هسته‌ای بسیار بیشتر است و هسته‌ها نیز ناهمگن (heterogenous) هستند. این نوع پردازنده‌ها نسل بعدی پردازنده‌ها را تشکیل می‌دهند. در شکل 1-1 نمایی از پنجره‌ی Task Manager در سیستمی با یک پردازنده‌ی چندین هسته‌ای نشان داده شده است.²

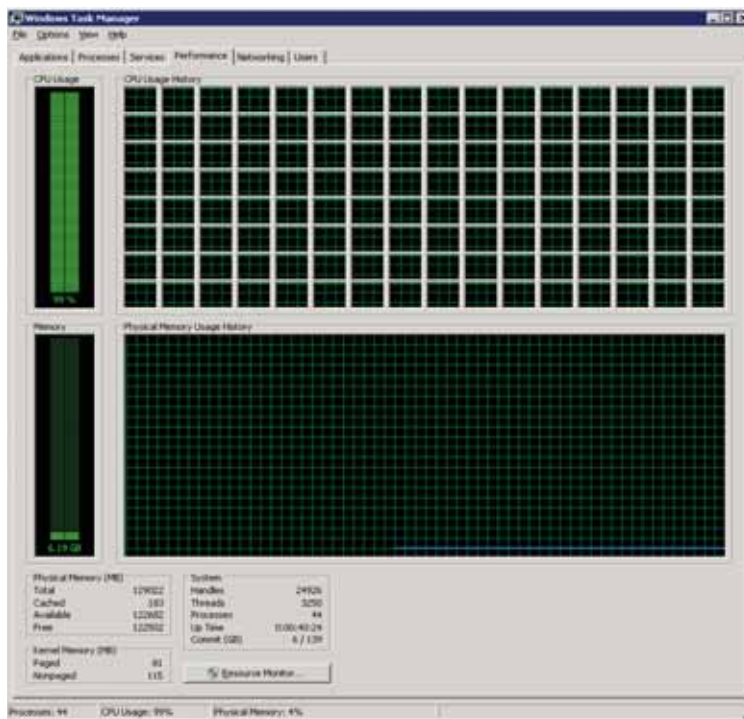
- **چند پردازنده‌ها (MultiProcessors)**

در این فناوری چندین پردازنده بر روی مادربورد گنجانده می‌شود. البته طبعاً مادربورد و کیس نیز بزرگ‌تر شده و در نتیجه مصرف برق آنها بیشتر است و گران‌تر هستند. این نوع سیستم‌ها فعلاً در حوزه‌ی سرویس‌دهنده‌ها رایج هستند و هنوز برای مصارف عمومی و خانگی محبوبیت پیدا نکرده‌اند. در حقیقت شرکت‌های تولید کننده از خودشان می‌پرسند چرا باید کامپیوتری بزرگ، سنگین، گران و پر مصرف تولید کنیم که تازه سرعت برنامه‌ها را عمدتاً به شرطی زیاد می‌کند که برنامه‌ها به شکل موازی نوشته شده باشد. لذا اگر کسی علاقه‌ای به خرید چنین سیستمی نداشته باشد عجیب نیست. اما شکی وجود ندارد که در سال‌های بعد این وضعیت تغییر خواهد کرد.

چیزی که بین تمام این فناوری‌ها مشترک است (البته به جز فناوری فرانخی) این است که در چنین سیستم‌هایی می‌توان برنامه‌ها را به طور واقعاً موازی اجرا کرد، نه این که اجرای هم‌روند آنها شبیه‌سازی شود. شبیه‌سازی اجرای

² منبع: <http://www.danielmoth.com/Blog/windows-7-task-manager-screenshot.aspx>

هم‌روند مکانیزمی بود که در سیستم‌های قدیمی و تک‌پردازنده انجام می‌شد. در واقع شما فکر می‌کردید برنامه به طور موازی کار می‌کند، در حالی که واقعاً یک برنامه‌ی ترتیبی بود.



شکل 1-1. پنجره‌ی Task Manager در سیستمی با یک پردازنده‌ی چندین هسته‌ای

وضعیت فعلی و آینده

واقعیت این است که چه دلمان بخواهد چه نخواهد سیر کامپیوترها به سمت چند پردازنده‌ها و پردازنده‌های چند هسته‌ای تغییر کرده و ما هم به عنوان برنامه‌نویس باید دیدگاه‌مان را عوض کنیم. چیزی که می‌توانم به شما بگویم این است که اگر از شیرجه زدن در این استخر واهمه دارید، حداقل در قسمت کم عمق کنار لبه‌ی استخر بنشینید و پایتان را در آب بیندازید. چون به هر حال باید در این استخر شنا کنیم! خبر خوب آن که مایکروسافت با معرفی TPL (کتابخانه‌ی اختصاصی برنامه‌نویسی موازی در .NET 4.0)، راه را برای ما کاملاً هموار کرده است.

معماری سیستم‌ها

تک پردازنده‌های چند هسته‌ای

در گذشته شرکت‌های تولید کننده‌ی تک پردازنده‌ها در طراحی پردازنده‌هایشان نکات زیادی در نظر می‌گرفتند تا هم مصرف برق کاهش پیدا کند، هم حرارت کمتری تولید شود و هم قابلیت‌های بهتری برای پردازش موازی فراهم شود تا در نهایت توان عملیاتی پردازنده³ افزایش پیدا کند. به عنوان مثال امروزه بیشتر پردازنده‌های مدرن قابلیت‌ی دارند که

³ CPU throughput