# TPL

TPL מפשטת את הדרך לעבוד עם קוד מקבילי וליצור תוכניות מקביליות (Parallelism או Concurrency).

החל מדוטנט 4, TPL זו הדרך המומלצת לתכנות מרובה תהליכים. צריך לזכור שלא כל קוד מתאים לפרלליזם. למשל, אם יש מעט איטרציות ללולאה או שבכל איטרציה יש לנו רק מעט קוד, לפעמים הסיבוכיות של הריצה המקבילית עלולה לגרום לתוכנית שלכם לרוץ לאט יותר, והיא כמובן תהיה קשה יותר לתחזוקה.

## Data Parallelism

כשרוצים לעבוד על סדרות/סטים כמו מערכים ואוספים, ורוצים לעשות פעולות בצורה מקבילית על יותר מאיבר אחד בסט, ניתן להשתמש בלולאות for ו foreach מיוחדות המוגדרות במרחב השמות: [System.Threading.Tasks](https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallel) . הן מאפשרות לנו לכתוב קוד שדי דומה לקוד שהיינו כותבים בלולאות רגילות, אבל היא מריצה את העיבוד על איברי המערך/אוסף מקבילית.

// Sequential version

foreach (var item in sourceCollection)

{

Process(item);

}

// Parallel equivalent

Parallel.ForEach(sourceCollection, item => Process(item));

בדוגמה אנחנו קוראים למתודה Process עבור כל איבר, זה קורה מקבילית.

בדוגמה הבאה נשתמש בלולאת Parallel.For לבדיקה מהירה של גודל תיקיה. גודל תיקיה היא נגזרת של גודל הקבצים שבה. כרגע בדוגמה לצורך הפשטות נניח שיש רק קבצים או שלא מעניין אותנו תתי התיקיות.

הפרמטר הראשון ב for הוא מאיזה index להתחיל, השני הוא עד איזה אינדקס לרוץ, והאחרון זה כמובן המתודה שיש להפעיל, צריך להעביר מתודה שמקבלת int ומתודה ה Parallel.For אחראית להעביר למתודה שלנו את האינדקס בכל איטרציה. היא תפעיל את המתודה שלנו מקבילית על מספר איברים בו זמנית.

long totalSize = 0;

var files = Directory.GetFiles(@"D:\");

Parallel.For(0, files.Length, i =>

{

FileInfo fi = new FileInfo(files[i]);

/\*\*\*\*\* The next line is not good since it may be updated by 2 treads in tsame time and caise wrong results \*\*\*\*\*\*/

//totalSize+=fi.Length;

/\*\*\*instead of building our own method that locks an object ew can use InerLocked\*\*\*\*/

Interlocked.Add(ref totalSize, fi.Length);

});

Task

אנחנו יכולים ליצור Task בכמה דרכים.

הגדרה ישירה

Task taskA = new Task( () => Console.WriteLine("Hello from taskA."));

// Start the task.

taskA.Start();

שימוש ב Task.Run

המתודה מורצת מיד

// Define and run the task.

Task taskA = Task.Run( () => Console.WriteLine("Hello from taskA."));

ניתן להשתמש גם במתודה הסטטית הבאה

Task.Factory.StartNew(() => { });

*המתודה Task.Run נוספה בהמשך, המתודה של TaskFactory מאפשרת יותר שליטה ויותר אופציות במקרים שאנחנו מתקדמים ורוצים להעביר כל מיני פרמטרים למערכת ש"ירמזו" לה איך לנהל את ה Task ואם לבצע את ה Task ב Thread הנוכחי או אחר ועוד כל מיני אופציות (שכשנגיע לצורך בחיים האמיתיים נלמד אותם, כרגע בשלב זה לא נכון לנו להתעכב עליהם.*

## העברת פרמטרים ל Task

//Here we are using a global vaiable

// Be awere who is updating it and if uou need a lock

var x = 0;

## var t = Task.Run(() => x + 3);

## החזרת ערך

/We can pass one object parameter

//We returning Task<string> in the below lines

var t1 = Task<string>.Factory.StartNew((o) => ("holy " + o), "cow");

//Implicitly returns Task<string>

var t2 = Task.Factory.StartNew((o) => { return "my holy " + o.ToString(); }, "cow");

## Task.Status

Taskיכול להיות במצבים שונים, כמו "מחכה לרוץ, בריצה, מבוטל, ועוד.

עברו על הסטטוסים האפשריים (F12 וכדו')

## Wait

t2.Wait();

Task.WaitAny(t1, t2);

Task.WaitAll(t1, t2);

## ChainingTasks

ניתן להריץ Taskולהגדיר שבסיומו ירוץ קוד אחר.

Task.Run מחזיר Task.

ל Task יש מתודה ContinueWith שגם היא בתורה מקבלת Action ומחזירה Task. המתודה ContinueWith מריצה את המתודה שהעברנו ל delegate בתוך Task ותחזיר את ה Task.

זה מאפשר לנו ראשית להריץ קטע קוד שתלוי בסיום של קוד אחר. שנית, בינתיים כמו ב Taskים אחרים קוד אחר יכול לרוץ.

## Asynchronous Programming

אם ה CPU שלנו מחכה לפעולה חיצונית, בינתיים ה Thread שלנו יכול להריץ עבודות אחרות. בגישה לרשת, לקבצים למסדי נתונים, פיתחו עבורנו תשתית שמשתמשת ב Taskים ומאפשרת להריץ קוד אחר כשהמעבד בהמתנה למשאב אחר. אם נשתמש בתשתית הזו, במתודות האלו, אז גם הקוד שלנו לא יחסום פעולות אחרות מלהתבצע כשאנחנו ממתינים והמעבד פנוי.

המתודות האלו מסתיימות במילה Async. למשל: HttpClient.GetStringAsync שמחזירה תוכן של דף מהרשת. המתודות האלו מחזירות Task שלו נוכל להמתין, או להירשם לפעולות שיקרו בסיומו. השימוש ב ContinueWith עם מתודות אלו, במקון ב Wait יאפשר לתהליך שלנו לבצע פעולות אחרות בזמן המתנה ל Task . פעולות שכן תלויות בתוצאות של המתודה ה"א-סינכונית" נגדיר אותם בתוך ה ContinueWith והם יקרו כשה Task של מתודת ה Async יסמן שהוא סיים את עבודתו. הבעיה היא שזו צורת כתיבה מורכבת ולא טבעית כל כך. היא לא מזכירה את התכנות הרגיל הסינכרוני שלמדנו עד כה לכתוב, והיא קשה לתחזוקה. לכן, נולד הנושא הבא שלנו, שימוש במילות מפתח async\await ולתת לקומפיילר להגדיר TASKים ו ContinueWith מאחורי הקלעים.

## Async & await keywords

נתבונן בשני המתודות שלהלן:

A picture containing text

Description automatically generatedהמתודות זהות, ה GetWebAsync כתוב כמעט כמו מתודה רגילה, רק עם הוספת async בהגדרת המתודה, ושימוש ב await בקריאה ל WebClient.GetStringAsync, מה שאומר תבצע את הפעולה בTaskנפרד, ותקרא להמשך הקוד שאחרי שורת ה Await בתוך בלוק של ContinueWith. כל העבודה הקשה עברה לאחריות הקומפיילר.

## ביטול TASK

המנגנון של TASKים תומכים בביטול TASK, אך הוא דורש מעורבות של המתודה שלנו (המתודה הפועלת) שה Task מפעילה. ניתן להעביר ל Start או ל Run "אסימון ביטול", שיאפשר לנו לעצור את פעולת ה Task ולבטל אותו, וזוב נזכור שפעולת הביטול, תצטרך להתבצע על ידינו בקוד ש נכתוב ב מתודה הפועלת . המנגנון תומך בהעברת סטטוס ה Task למצב "ביטול מוצלח" או "ביטול שלא הצליח". כמו"כ המנגנון מספק לנו את שיטה אחידה לבצע ביטולים כאלו.

Task t1;

CancellationTokenSource cts=new CancellationTokenSource();

CancellationToken ct;

ct = cts.Token;

t1 = Task.Run(() =>

{

for (int i = 0; i < 5000; i++)

{

if (ct.IsCancellationRequested)

return;//Status of task will be RanToCompletion

Debug.WriteLine(i);

}

},ct);

//פה מבוצעת בקשת ביטול

//הסטאטוס של הטוקן עובר למצב - התבקש ביטול

cts.Cancel();

בדוגמה הזו ביקשנו Return וזה בסדר אם כך נרצה לממש, אבל סטטוס ה Task יהיה "סיים בהצלחה".

לעומת זאת, אם היינו כותבים

if (ct.IsCancellationRequested)

ct.ThrowIfCancellationRequested();

סטטוס ה Task היה הופך למבוטל. כשתנסו את הקוד הזה במצב פיתוח, זה יעצור לכם את התוכנית, אבל אם נתפוס את השגיאה אחרי ה Task ונתעלם מהשגיאה, כי היא מייצגת מצב תקין, אז כשנבדוק את הסטאטוס של ה Taskהוא יהיה Canceled.

שמתי את זה באפור, כי אני ממליץ לדלג על שיטה זו בשלב זה. לקרא, לדעת שקיים, ולהתקדם!