C#の高速化入門

Write by 森理 麟

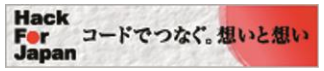
非同期勉強会

Myself

- 森理 麟(@moririring)
- 職業：ゲームプログラマ
- HP : moririringのHP
- ツール : VagrantWin ←NEW!
- Microsoft MVP for Visual C#(2013.01 –)

My Community

- VSハッカソン倶楽部 + Visual Studio勉強会
- C++テンプレート完全ガイド読書会
- Unityクリエイターズ
- IT英語勉強会
- ぼちぼちぼっち開発



アジェンタ

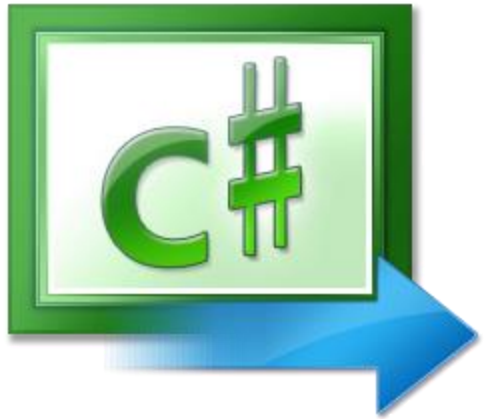
- はじめに
- キーワード説明
  - プロセス
  - スレッド
  - 同期/非同期
  - コア
- 高速化
  - ターンアラウンドタイム
  - スループット
  - レスポンスタイム
- おわりに

first

はじめに

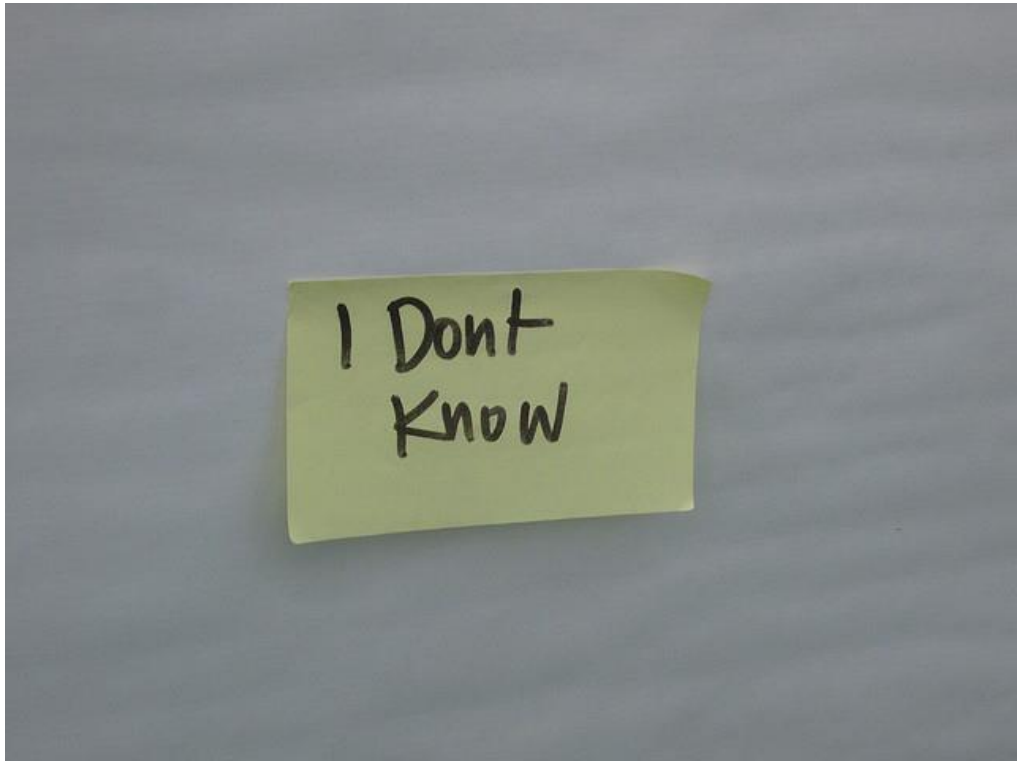
はじめに

- 僕がC#を始めたのは非同期プログラムを組みたかったからです。



はじめに

- もっとも当時の僕は非同期という言葉は知りませんでした。



はじめに

- 動き出してからも止めたい。そのためにはスレッドを使えば良いのかな？でした。

はじめに

- ただあまり詳しくない僕はマルチスレッドは高速化するための手法だと思っていました。

はじめに

- 非同期？高速化？マルチスレッド？はじめにこの当たりのキーワードを説明します。
  - はじめにの方が長い？

Process

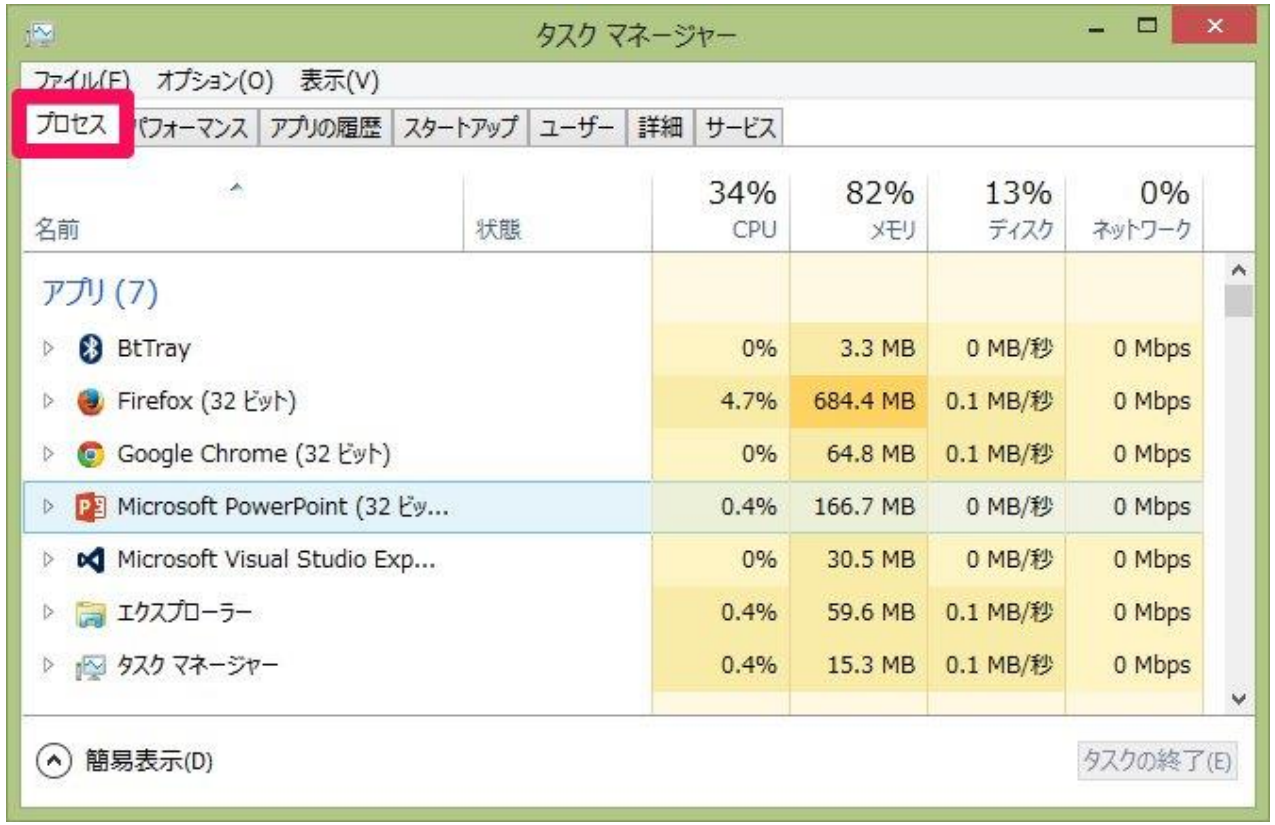
プロセス

プロセス

- プロセスとは実行しているプログラムの単位

プロセス

- 実行中のプロセスはタスクマネージャで確認出来る



プロセス

- プロセスは実行に必要なものをまとめて格納するコンテナ

プロセス

- プロセスはタスクとかジョブとも言う。タスクマネージャというぐらいなので。

プロセス

- C#プログラマとしては他の実行ファイルを起動する時にProcessを使う。超使う。

- `Process.Start("notepad.exe");`

Thread

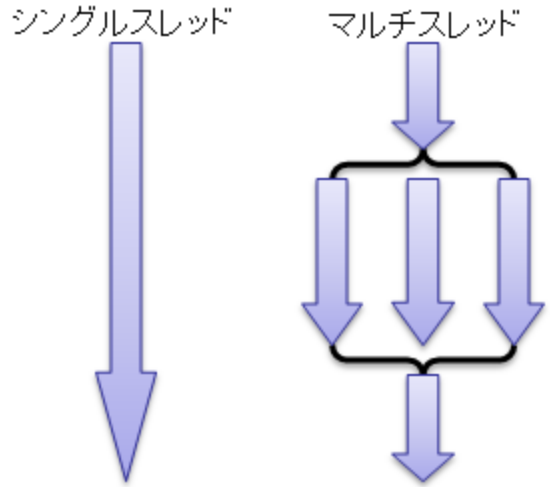
スレッド

スレッド

- スレッドとはプロセスの最小の実行単位（因みにThreadは糸のこと）

スレッド

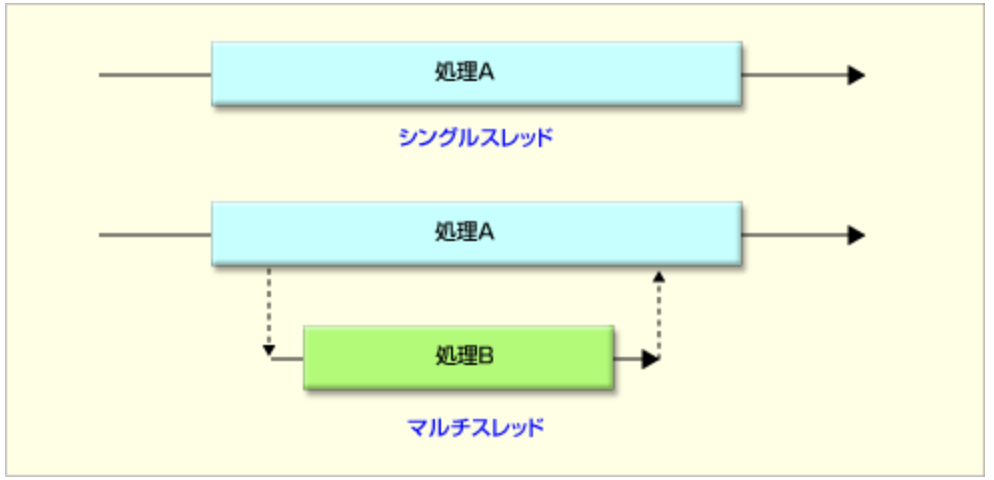
- １つのプロセスが複数のスレッドを持つことが出来る。これをマルチスレッドという。



- 引用:++C++; // 未確認飛行 C マルチスレッド (http://ufcpp.net/study/csharp/sp_thread.html)

スレッド

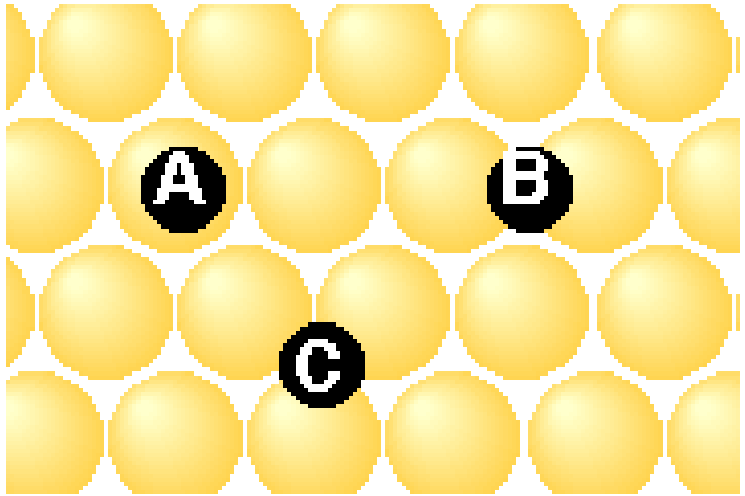
- マルチスレッドは同時に処理を実行出来る



- 引用:連載.NETマルチスレッド・プログラミング入門 (http://www.atmarkit.co.jp/ait/articles/0503/12/news025.html)

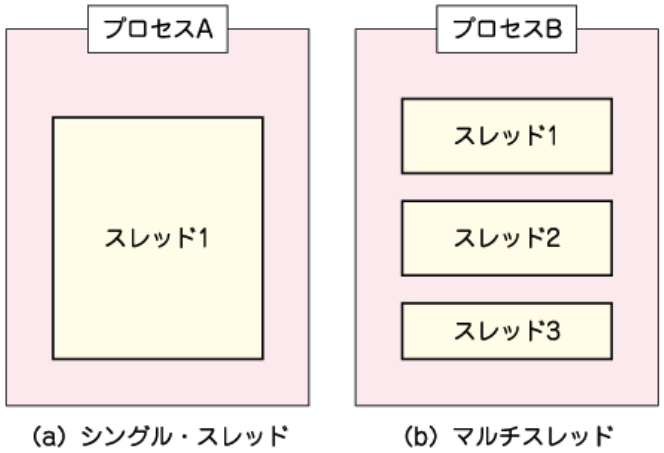
スレッド

- スレッドが集まってプロセスを構成する。プロセスもスレッドの一種。 原子と分子、フォルダとファイルのようなニュアンス。



スレッド

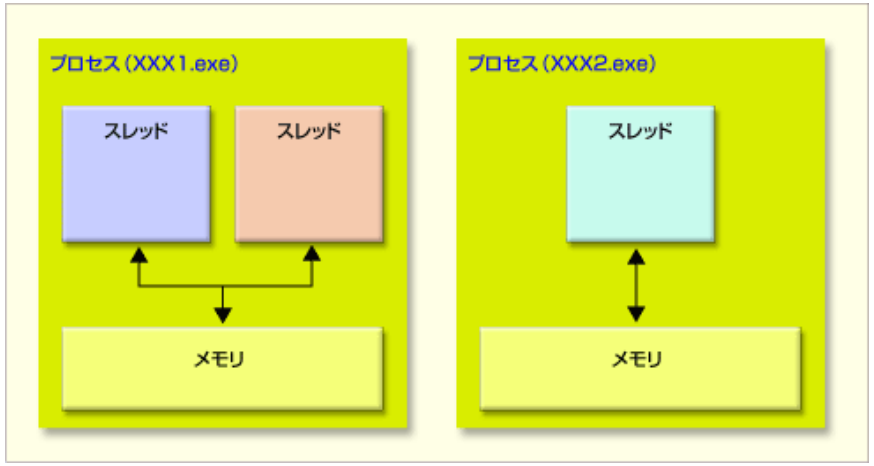
- マルチプロセスも同じように同時に実行できる



- 引用:組み込み分野における「マルチプロセッサ」とは　―― 多機能・低消費電力の要求にこたえるための技術的要素と課題 (http://www.kumikomi.net/archives/2005/09/15multic.php?page=13)

スレッド

- 違いはメモリ。プロセス間ではメモリは共有されない。スレッド間では共有される。



- 引用:連載.NETマルチスレッド・プログラミング入門 (http://www.atmarkit.co.jp/ait/articles/0503/12/news025.html)

まとめ

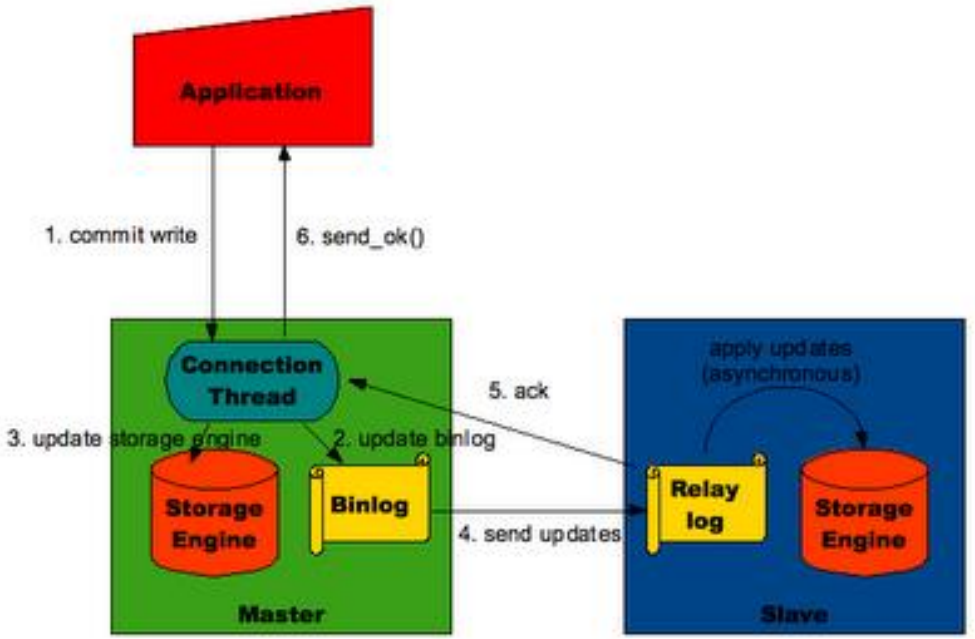
- プロセスはプログラムの実行単位
- スレッドはプロセス内に作られる処理の単位
- 1つのプロセスは1以上のスレッドを持つ
- スレッドもプロセスも複数持てて、同時に実行できる

synchronous/ asynchronous

同期/非同期

非同期

- プログラムの処理には同期と非同期がある



非同期

- 処理をして結果が返ってくるまで待つのは同期。リレーは同期。

非同期

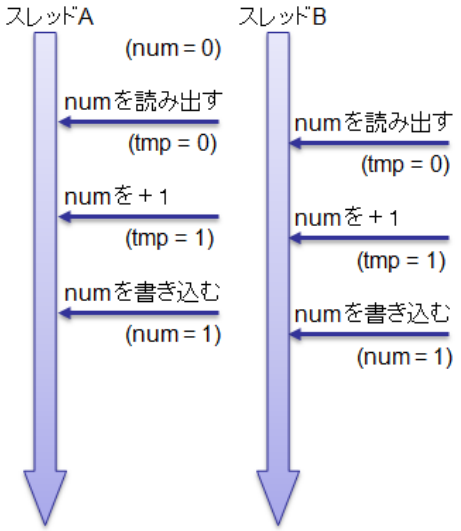
- 処理を待たずに次の処理を実行するのが非同期。玉入れは非同期。

非同期

- スレッドと同期、非同期は直接は関係ない

非同期

- しかしマルチスレッドは、スレッド間が常に非同期。



- 引用:++C++; // 未確認飛行 C マルチスレッド (http://ufcpp.net/study/csharp/sp_thread.html)

非同期

- つまりマルチスレッドは非同期プログラムと親和性が高い

Core

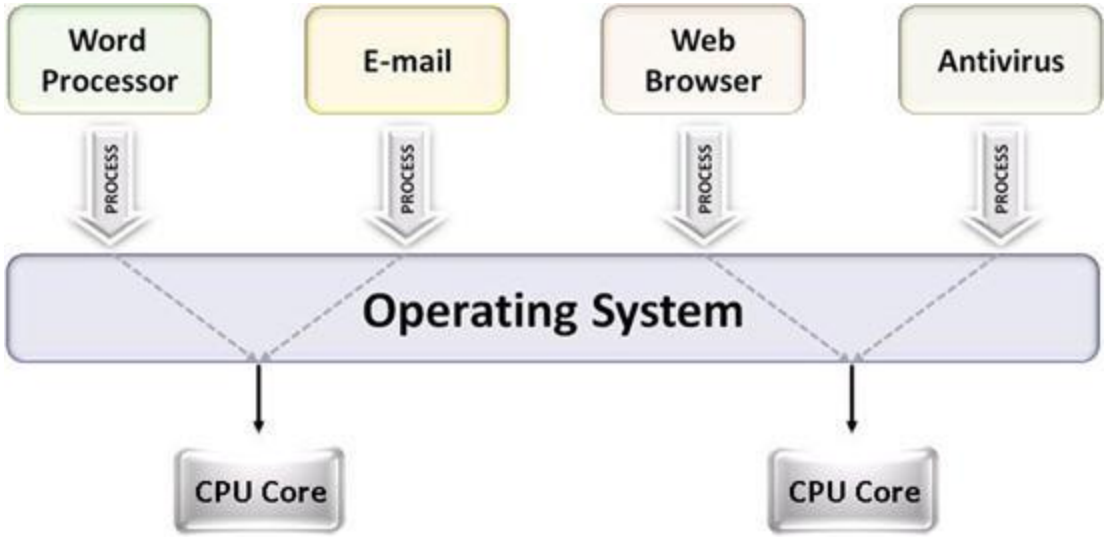
コア

コア

- コアとはCPUの中心部分であり、実際に処理を行うところ

コア

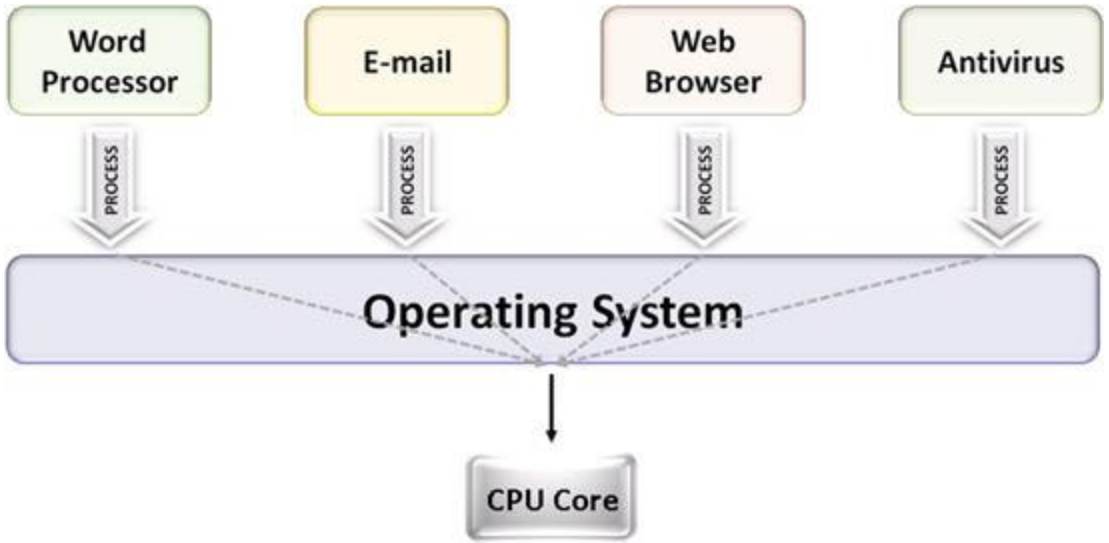
- スレッドやプロセスはコアを複数使用できる。マルチコア



- マルチスレッドとマルチタスクの違い (http://www.ni.com/white-paper/6424/ja/)

コア

- 但しコアが一つでも、マルチスレッドは可能。どうやるか？



- マルチスレッドとマルチタスクの違い (http://www.ni.com/white-paper/6424/ja/)

コア

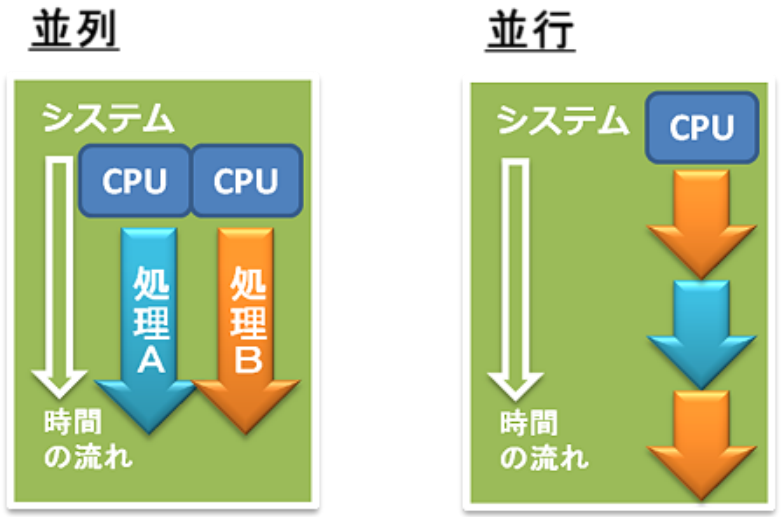
- こう

コア

- マルチスレッドにしても処理が速くなるとは限らない。むしろ遅くなることもある。

コア

- シングルコアマルチスレッドは並行処理、マルチコアマルチスレッドは並列処理。



- 並行と並列について-並行コンピューティング技法-を読んで (http://www.m-tea.info/2011/03/concurrent-parallel-01.html)

まとめ

- プロセス、スレッド
- 同期、非同期
- コア
- 並列、並行
- ↓
- マルチスレッドによる非同期
- マルチコアによる並列処理

高速化

高速化

- 僕は高速化って「地球にやさしい」ぐらい矛盾した言葉だと思っています。

高速化

- だってCPUは本来超高速です。3GHzのCPUなら1秒間に30億回動きます。
- そこにプログラマがプログラムで負荷をかけまくって、1万回のループで10万回の計算をさせたりして遅くします。
- そうやって自分で遅くさせたCPUを、自分で遅くなくするのが高速化です。

turnaround time

ターンアラウンドタイム

ターンアラウンドタイム

- ターンアラウンドタイムとは
  処理にかかる総時間のこと

ターンアラウンドタイム

- 最初の高速化はターンアランドタイムの短縮

ターンアラウンドタイム

- この高速化で最も重要なのが計測

ターンアラウンドタイム

- 人間の感覚で速い遅いを判断しては駄目

ターンアラウンドタイム

- ストップウォッチを使ってちゃんと計測しよう

ターンアラウンドタイム

- C#でもストップウォッチ

```
var sw = Stopwatch.StartNew();
ShallowWork();
Console.WriteLine(sw.Elapsed);
sw.Reset();
sw.Start();
DeepWork();
Console.WriteLine(sw.Elapsed);
sw.Stop();
//00:00:00.0026507
//00:00:05.0886134
```

- 但し計測だけならストップウォッチよりプロファイラーの方が便利。（指摘により加筆。Thanks to kkamegawa）

ターンアラウンドタイム

- 大体はパレートの法則(90:10の法則)が成り立つ

- ヴィルフレド・パレート

ターンアラウンドタイム

- 10%を絞り込めたら、そこの部分を高速化
- IOアクセスの回数を減らしたり、アルゴリズムを見直したり、事前に計算してメモリに乗せたり。
- C#なら16byte以下のclassをstructにしたり、~~キャストが重いのでasを使ったり~~（次ページに削除理由）、文字列の連携にはStringBuilderを使ったり

前ページの取り消し理由

- 「キャストが重いのでasを使う」と書いた所、適当ではないとの指摘を受ける。 (Thanks to isishizuka!)
- 例外が発生するケースではキャストよりasを使った方が高速化だけど、10%に絞り込んだ時にキャストをasに直すというのは適当ではないため（100%確実に型変換ができる場合はキャストの方が高速）取り消しました。

ターンアラウンドタイム

- 奇抜なテクニックで速くなることはあまりないかも
- 地味で手間はかかるが、トライアンドエラーを繰り返す
- 学問には王道しかない

↑引用:森博嗣著「喜嶋先生の静かな世界」

ターンアラウンドタイム

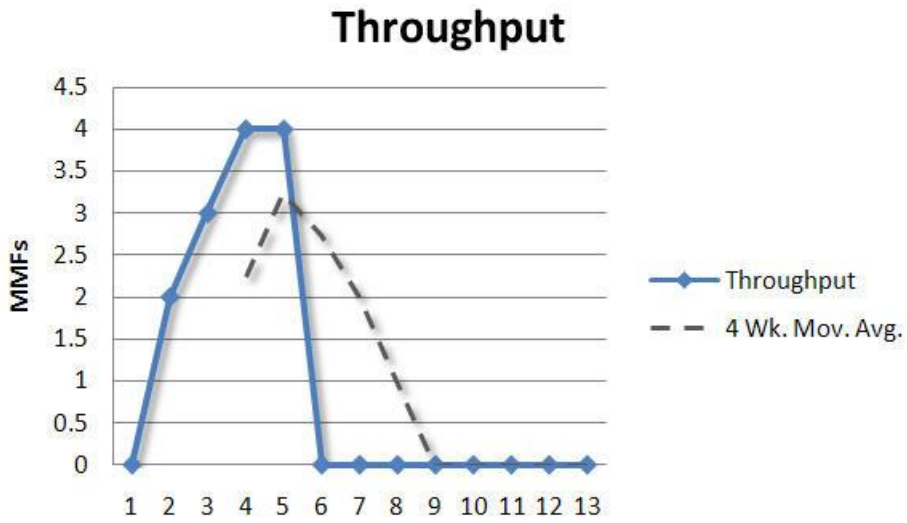
- 同じ場所が再度遅くなることもよくある
- 計測結果は実際に動いている環境に表示したり、ログを残したり見える化すると良い

throughput

スループット

スループット

- スループットとは時間当たりで処理出来る量のこと。



スループット

- 次の高速化はスループッドの増加

スループット

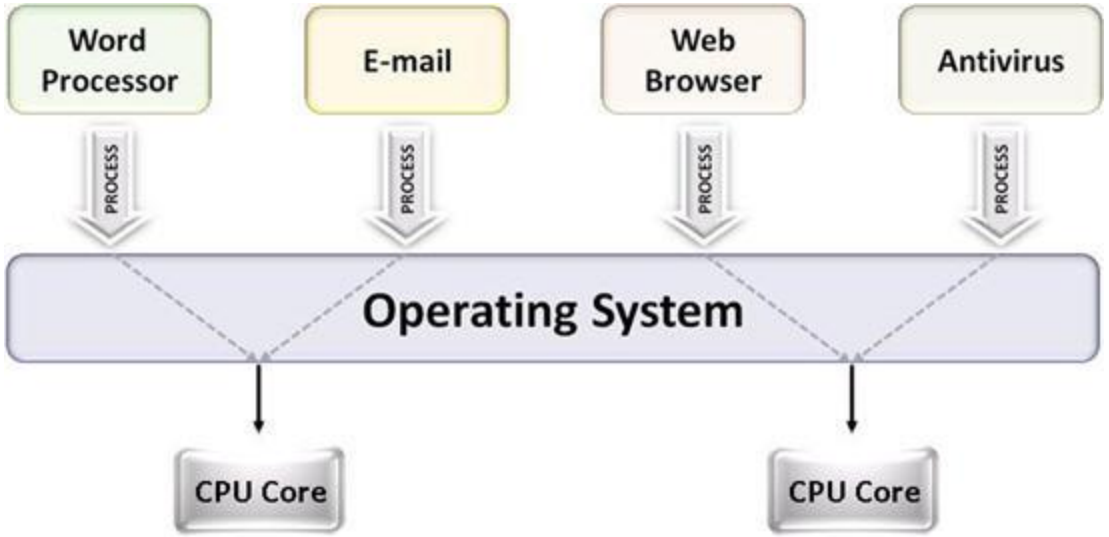
- スループットを増やすにはコアを増やすこと。

スループット

- 8分かかったものを4人で処理すれば2分になる。仕事量が8分であることには変わりない。

スループット

- シングルコアでの並行処理では速くならない。マルチコアでの並列処理をする



スループット

- マルチコアプログラムを初めて組んだ時僕はとても驚きました。

スループット

- 地味で手間のかかる高速化をしていないのに、びっくりするぐらい速くなった

スループット

- DEMO1

```
//通常for
for (int i = 0; i < 100; i++)
{
    DeepWork();
}
//並列実行Paraller.For
Parallel.For(0, 100, i =>
{
    DeepWork();
});
```

スループット

- ただ、マルチコアプログラムは実際難しい点も多い。

- indexは順番不定
- UIスレッド
- 排他制御

スループット

- 注意点はしっかり押さえよう！
- C# によるプログラミング入門
  並列処理ライブラリ　<- 岩永さんとか
- この後のbiacさんのセッションでも…！
- MSDN - データとタスクの並列化における注意点
- 並列コンピューティングとは何か

ポイント

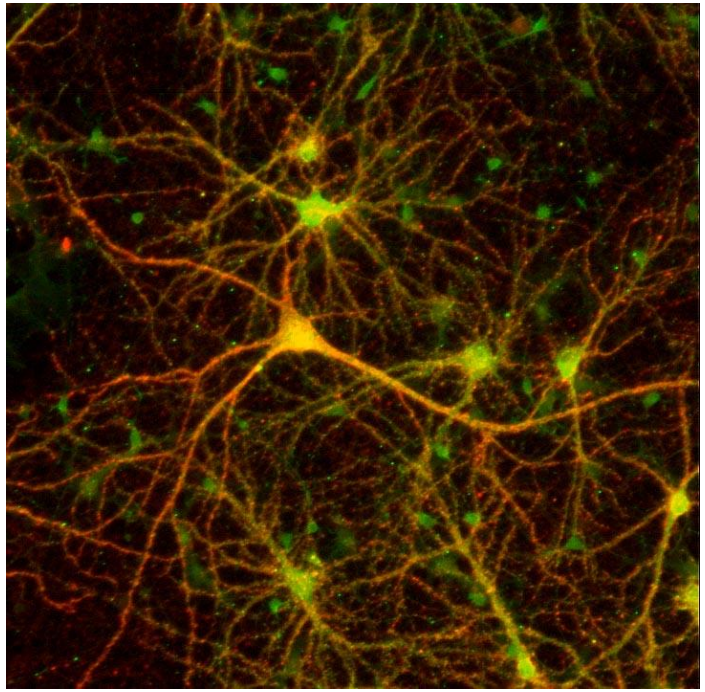
- ターンアラウンドタイムを減らし、スループッドを増やせば速くなる
- しかし、速くなってもユーザーは最終結果がファーストコンタクト
- 分かりやすく言うと1分を15秒にしても皆(隣も!) 遅いと言う

Response time

レスポンスタイム

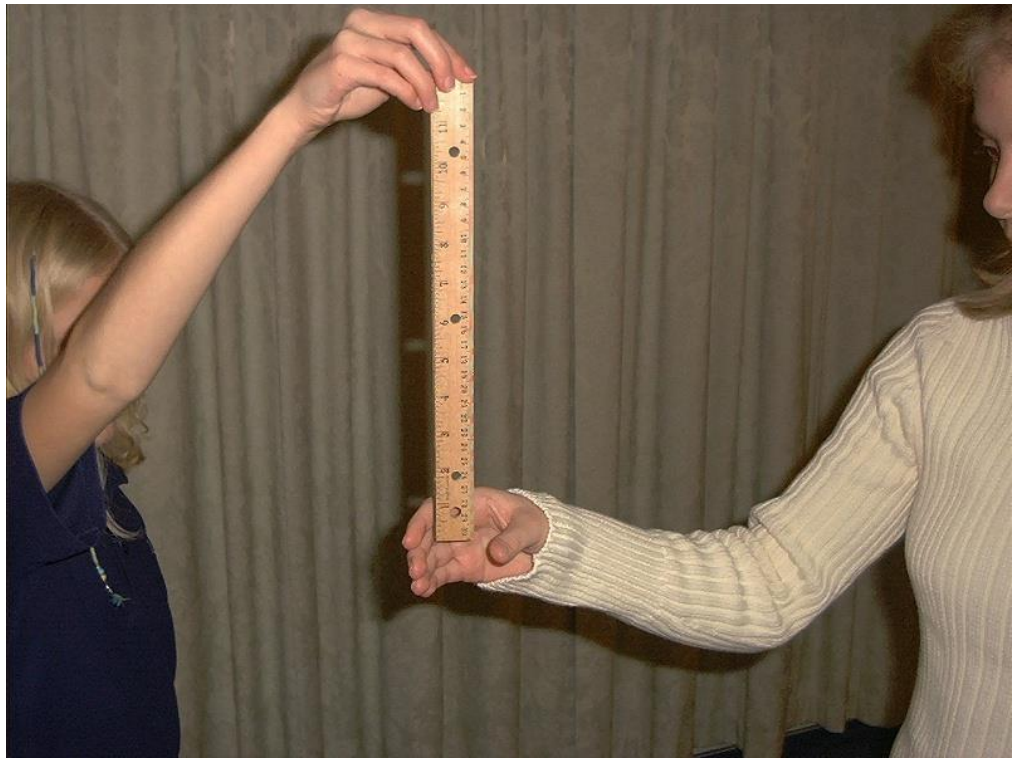
レスポンスタイム

- レスポンスタイムとはアクションに対してリアクションが返ってくるまでの時間。



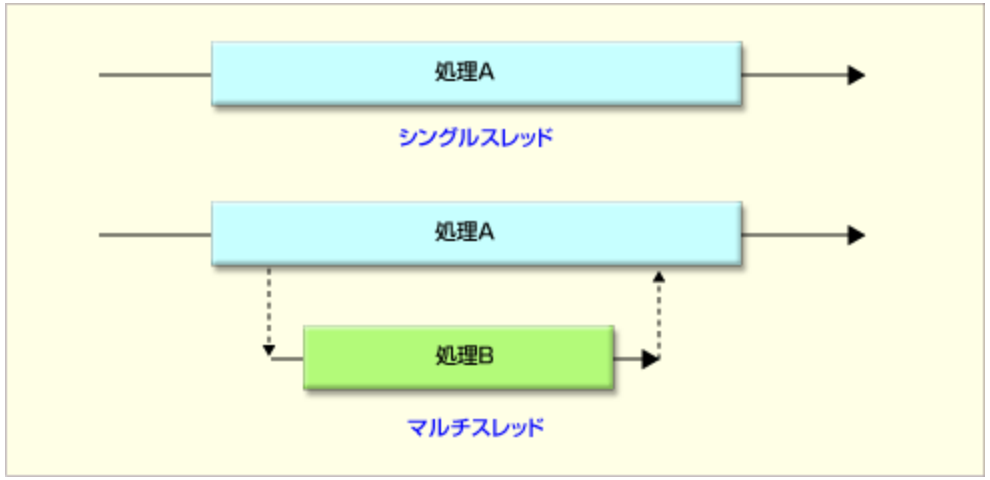
レスポンスタイム

- 最後の高速化が、レスポンスタイム、反応の速さの高速化。



レスポンスタイム

- レスポンスタイムを向上するにはマルチスレッドによる非同期プログラミング



レスポンスタイム

- 非同期にするとフリーズしなくなる。DEMO2

```
Task.Run(() =>
{
    for (int i = 0; i < 100; i++)
    {
        DeepWork();
    }
});
```

レスポンスタイム

- しかし待つことは変わらないケースが殆ど。

レスポンスタイム

- そこで、待つのが気にならないコントロールでのUI/UX表現が必須。

レスポンスタイム

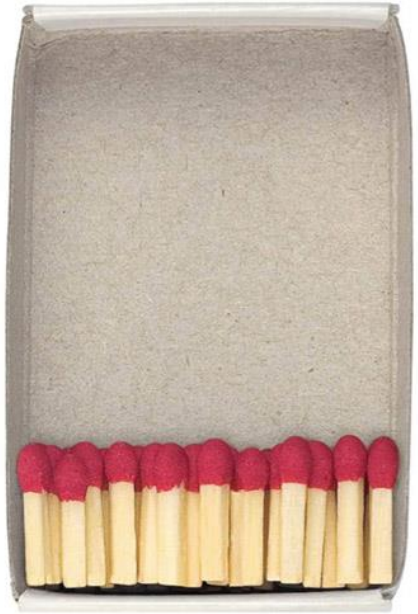
- コントロールの中で数少ないボタンが持っているUI/UXの特性があると思っています。
- コンボボックスが次点？

レスポンスタイム

- アクションに対しての反応が自分自身でない(ことが多い)ので即時でなくても待ちやすい。

レスポンスタイム

- 他のコントロールはリアクションが自分自身のため即座でないと使いにくさを感じる。
- DEMO3



レスポンスタイム

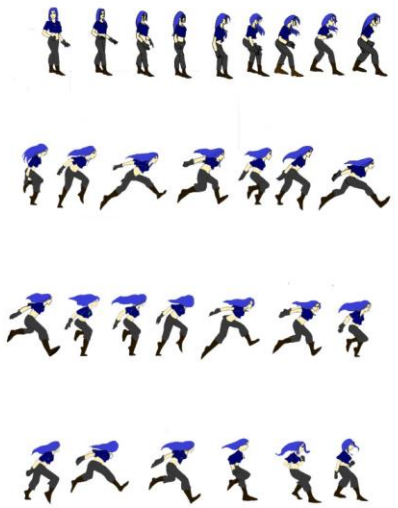
- ボタン自体もリアクションとしてEnabledをoffにすると分かりやすいし、誤動作防止。
- DEMO4

レスポンスタイム

- ボタン表示オフだけより何をしているのかを文字で書くだけでも印象が違う。
- DEMO5

レスポンスタイム

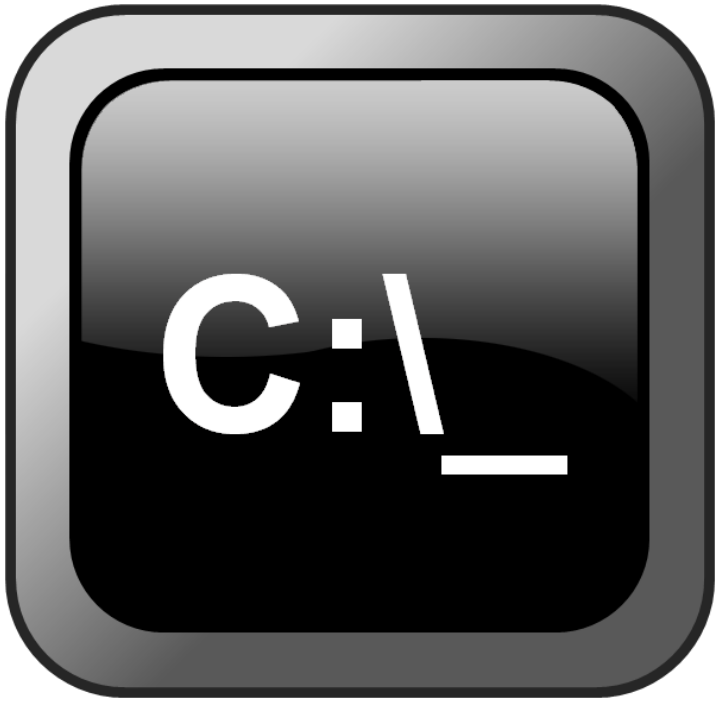
- 特に重要なのが動き。動いているコントロールがあれば待ちやすい。



- Low Wai Yin's Portfolio:http://lwaiyin.wordpress.com/2011/02/25/2d-run-animation/

レスポンスタイム

- コマンドラインでもキュレットが点滅しているので待てる



レスポンスタイム

- ここで大問題。マルチスレッドにするとコントロールへの書き込みが出来ない。

レスポンスタイム

- 対応するにはWPFならDispacherやWinFormならInvoke。もしくはTaskSchedulerを使う
- DEMO6,7

レスポンスタイム

```
var uiTask =
TaskScheduler.FromCurrentSynchronizationContext();
await Task.Run(() =>
{
    for (int i = 0; i < 100; i++)
    {
        DeepWork();
        new Task(() => progressBar1.Value = i).Start(uiTask);
    }
});
```

レスポンスタイム

- さらにプログレスバーにキャンセル処理。やっと最初の下りの実現。
- DEMO8

レスポンスタイム

```
var uiTask = TaskScheduler.FromCurrentSynchronizationContext();
await Task.Run(() =>
{
    for (int i = 0; i < 100; i++)
    {
        DeepWork();
        if (_cancelToken.IsCancellationRequested) break;
        new Task(() => progressBar.Value = i).Start(uiTask);
    }
});
private CancellationTokenSource _cancelToken = new
CancellationTokenSource();
private void cancelButton_Click(object sender, EventArgs e)
{
    _cancelToken.Cancel();
}
```

まとめ

- ターンアラウンドタイムを減らす。泥臭いけど王道。
- スループットを増やす。マルチコアによる並列プログラム。
- レスポンスタイムを速く。マルチスレッドによる非同期プログラム。
- レスポンスタイムを速くして、待つのが気にならないUIを作る

end

おわりに

おわりに

- 最後に高速化を比喩で。
- 高速化はダイエットに似ています。

DEBU!

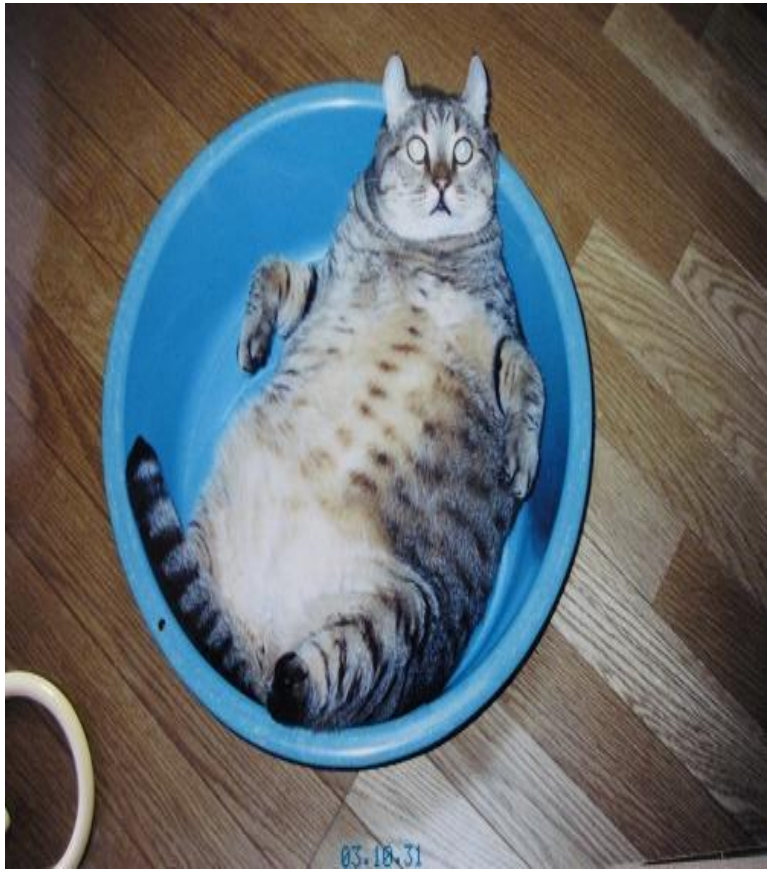
- 太ります！

CHECK!!

- 測ります！！

YASE!!!

- 痩せます！！！



REBOUND!!!!

- リバンウンドします！！！！

BEAUTIFUL!!!!!

- ごまかします！！！！！

参考(一部)

- 連載.NETマルチスレッド・プログラミング入門
- マルチスレッドとマルチタスクの違い
- ++C++;// 未確認飛行 C　マルチスレッド
- 意味の違いがわかる？ タスクとプロセスとスレッド
- 避けて通れない「非同期処理」を克服しよう
- ++C++;// 未確認飛行 C　並列処理ライブラリ
- C# 5.0＆VB 11.0新機能「async／await非同期メソッド」入門
- 並行と並列について-並行コンピューティング技法-を読んで
- 並列プログラミング入門！＆おさらい！
- Taskを使って非同期処理
- C#5.0のasync/awaitによる非同期処理の使い方
- 非同期処理でUIスレッドを操作する方法
- とあるコンサルタントのつぶやき マルチスレッド Windows フォームアプリケーションの開発
- DOBON.NETより高い精度で時間を計測する
- VC++まわりの非同期処理
- 増補改訂版Java言語で学ぶデザインパターン入門マルチスレッド編
- @IT, Wikipedia, IT用語辞典, StackOverFlow,

おわりに

- 発表するとめちゃめちゃ勉強になりますよ。皆さんもぜひ次回しゃべってください！

おわりに

- ご清聴ありがとうございました

今日のソース

- https://github.com/moririring/SpeedupDemo