GRAND CENTRAL DISPATCH (GCD)

Kaê Angeli Coutinho

O QUE É GCD?

- É uma tecnologia desenvolvida pela Apple para melhorar o desempenho de aplicativos usando processadores multi-cores (multi-núcleos)

- Em outras palavras, é uma API com recursos de multiprocessamento que gerencia núcleos do processador, controle e desempenho de tarefas paralelas (síncronas ou assíncronas)

CURIOSIDADES

- Escrito em C
- Open source
- Vem do nome “Grand Central Terminal”, a principal estação de trem de Nova Iorque
- Muitas bibliotecas utilizadas como SDWebImage e AFNetworking só foram possíveis de serem implementadas, através da intensa utilização do GCD

COMO O GCD PODE MELHORAR MEU APLICATIVO?

- A melhoria de desempenho é feita através da mudança automática (gerenciado pelo GCD) do núcleo de processamento atual

- O GCD é inteligente suficiente para detectar um núcleo mais ocioso e enviar todo o processamento para lá

- Entre outras palavras, lidamos com threads (tarefas em paralelo) mais rápidas e com mais confiabilidade

APLICAÇÃO PRÁTICA

- A palavra "automática" foi bastante citada, porém, o GCD não está habilitado para todas os blocos de código de um projeto de imediato, é necessário chama-lo através de funções

- O GCD compreende um número enorme de funções e tarefas. Estas estão entre as mais importantes

```
void dispatch_main()
```

- Executa um bloco de código na thread principal (thread geral do sua aplicação)

- Muito utilizado quando precisamos atualizar a interface gráfica, tendo em mente que, quando mandamos um bloco de código ser executado em paralelo em outra thread distinta da thread principal, a interface gráfica fica inacessível (apenas quando assíncrono) à essa thread em discussão

```
void dispatch_once(dispatch_once_t * predicate,dispatch_block_t block)
```

- Executa um bloco de código uma vez, e somente uma vez para todo o ciclo de vida de uma aplicação, sem contar que é thread-safe (muito confiável)

- Muito utilizado em serviços de autenticação e em padrões de projeto (Singleton)

```
void dispatch_sync(dispatch_queue_t queue, dispatch_block_t block)
```

- Executa um bloco de código em outra thread (gerenciado pelo GCD) de maneira síncrona, ou seja, quando a função é chamada, todo o trabalho (bloco de código enviado) será feito antes da função retornar

- Muito utilizado quando temos blocos complexos e robustos porém que possam ser executados de maneira rápida ou até mesmo quando queremos melhor gerenciar o uso da thread principal, mandado trabalho para alguma outra thread mais ociosa

```
void dispatch_async(dispatch_queue_t queue, dispatch_block_t block)
```

- Executa um bloco de código em outra thread (gerenciado pelo GCD) de maneira assíncrona, ou seja, quando a função é chamada, ela retorna imediatamente, todo o trabalho (bloco de código enviado) será feito em uma fila assíncrona ao tempo da sua aplicação

- Muito utilizado quando temos dados quaisquer que necessitam ser baixados da internet como imagens, JSONs, XMLs, YAMLs, entre outros

```
void dispatch_get_global_queue(long priority, unsigned long flags)
```

- Retorna uma thread paralela bem avaliada, a partir dos argumentos passados

```
void dispatch_get_main_queue()
```

- Retorna a thread principal (utilizada pela sua aplicação)

AINDA NÃO ENTENDI

- Dentre outras palavras, GCD é uma ferramenta que irá ajudar-nos a fazer tarefas mais complexas e interessantes sem travarmos nossa aplicação por conta do uso excessivo de um único núcleo no processador, logo que estaremos usando vários núcleos do mesmo processador para contornarmos tal problema

- Se você ainda tem dúvidas quanto a este tópico, sugiro pesquisarem mais sobre o assunto nos links abaixo

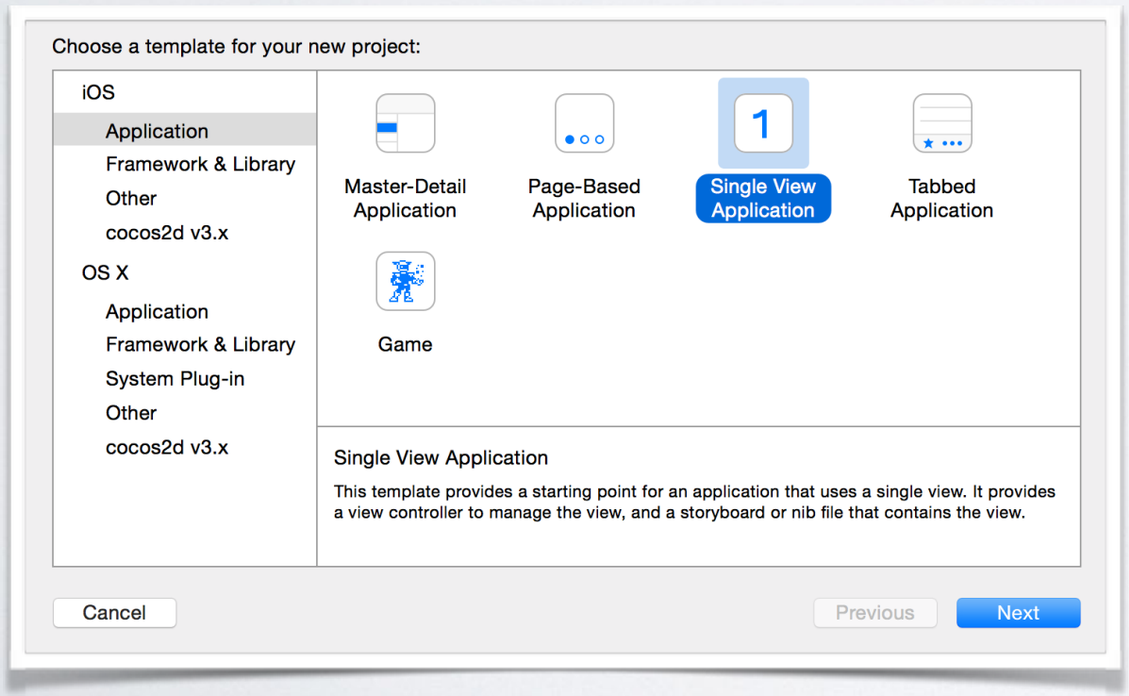
    - https://developer.apple.com/library/mac/documentation/performance/reference/gcd_libdispatch_ref/Reference/reference.html

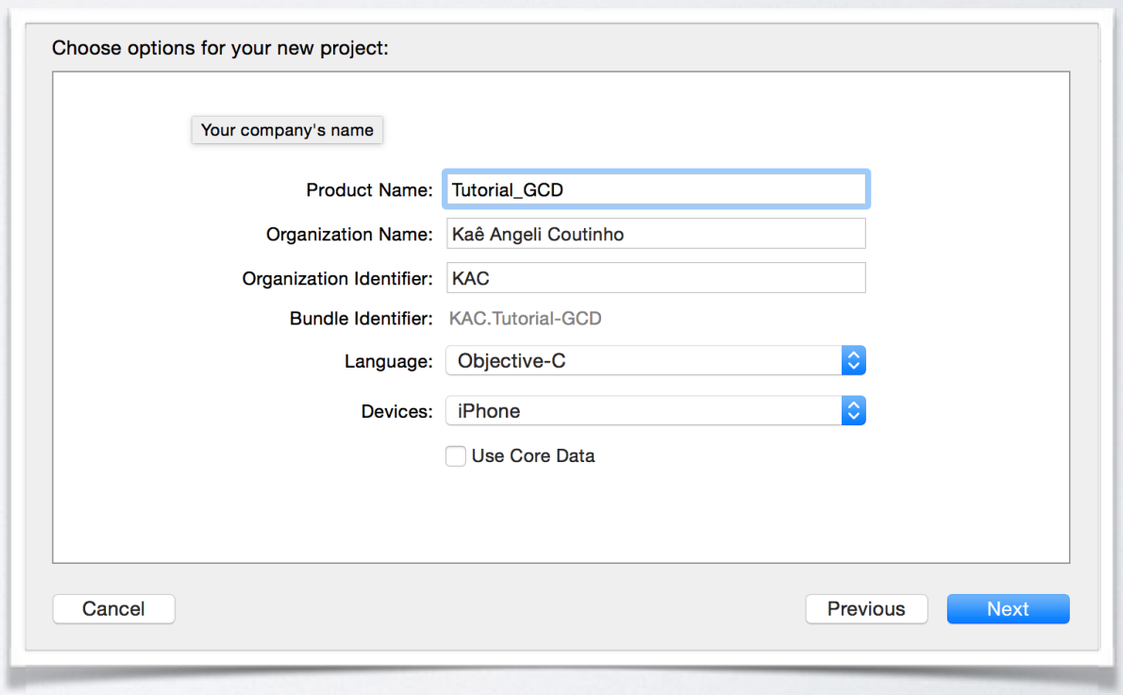
    - http://www.raywenderlich.com/4295/multithreading-and-grand-central-dispatch-on-ios-for-beginners-tutorial

Kaê Angeli Coutinho

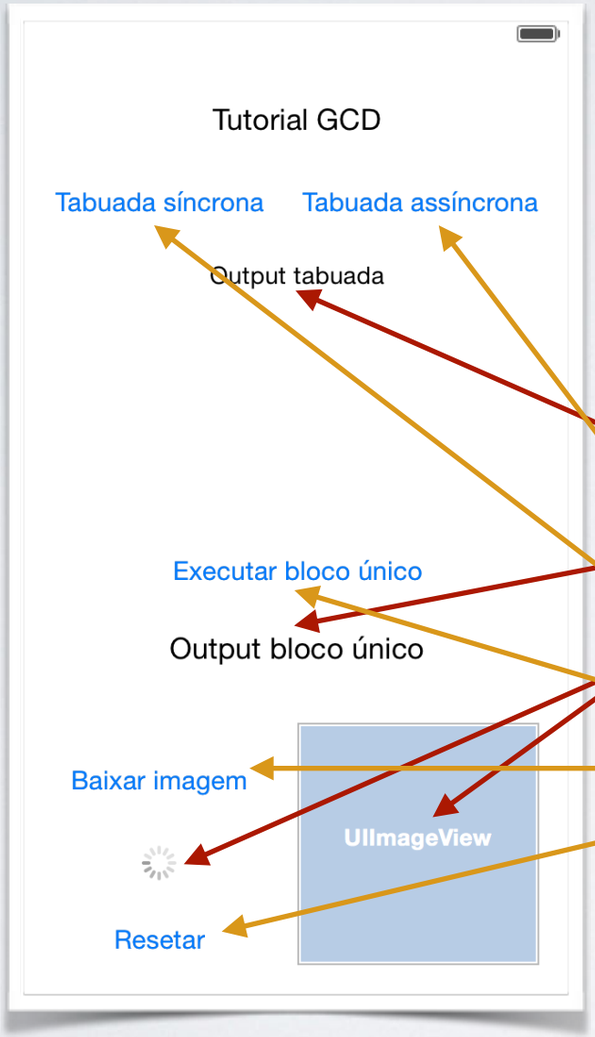
MÃOS À OBRA

- Iremos criar agora, um projeto no Xcode usando o GCD para:
    - Baixar uma imagem assíncrona da internet
    - Executar cálculos síncronos e assíncronos
    - Executar um bloco de código uma vez, e somente uma vez durante todo o ciclo de vida da aplicação





- Começaremos desenvolvendo a interface gráfica e atrelando-a com o ViewController.h




```
1  //
2  //  ViewController.h
3  //  Tutorial_GCD
4  //
5  //  Created by Kaê Coutinho - BEPiD on 3/17/14.
6  //  Copyright (c) 2014 Kaê Coutinho. All rights reserved.
7  //
8
9  #import <UIKit/UIKit.h>
10
11 @interface ViewController : UIViewController
12
13 @property (nonatomic,weak) IBOutlet UITextView * outputTabuadaTextView;
14 @property (nonatomic,weak) IBOutlet UILabel * outputBlocoUnicoLabel;
15 @property (nonatomic,weak) IBOutlet UIImageView * outputImagemImageView;
16 @property (nonatomic,weak) IBOutlet UIActivityIndicatorView * activityIndicator;
17
18 -(IBAction)tabuadaSincrona:(UIButton *)sender;
19 -(IBAction)tabuadaAssincrona:(UIButton *)sender;
20 -(IBAction)executarBlocoUnico:(UIButton *)sender;
21 -(IBAction)baixarImagem:(UIButton *)sender;
22 -(IBAction)resetar:(UIButton *)sender;
23
24 @end
```

- IBOutlet
- IBAction

- Vamos agora implementar todos os IBActions

```objc
-(IBAction)tabuadaSincrona:(UIButton *)sender
{
    [outputTabuadaTextView setText:@"SÍNCRONO"];
    // Abertura de processo paralelo síncrono
    // A constante "DISPATCH_QUEUE_PRIORITY_DEFAULT" representa o valor "0" e significa que irá retornar a thread mais bem avaliada e
        ociosa naquele momento de sua chamada
    // O segundo parametro "0" significa a não adição de flags para tal função, apenas o padrão
    dispatch_sync(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,0),^
    {
        // Algoritmo de tabuada do 0 -> 10
        for(NSInteger numeroAtual = 0; numeroAtual <= 10; numeroAtual++)
            for(NSInteger iteracao = 0; iteracao <= 10; iteracao++)
                // Atualiza o UITextView com as informações atuais
                [outputTabuadaTextView setText:[NSString stringWithFormat:@"%@\n%ld x %ld = %ld",[outputTabuadaTextView text],(long)
                    numeroAtual,(long)iteracao,(numeroAtual * iteracao)]];
    });
}
```

```objc
-(IBAction)tabuadaAssincrona:(UIButton *)sender
{
    [outputTabuadaTextView setText:@"ASSÍNCRONO"];
    // Abertura de processo paralelo assíncrono
    // A constante "DISPATCH_QUEUE_PRIORITY_DEFAULT" representa o valor "0" e significa que irá retornar a thread mais bem avaliada e
        ociosa naquele momento de sua chamada
    // O segundo parametro "0" significa a não adição de flags para tal função, apenas o padrão
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,0),^
    {
        // Algoritmo de tabuada do 0 -> 10
        for(NSInteger numeroAtual = 0; numeroAtual <= 10; numeroAtual++)
            for(NSInteger iteracao = 0; iteracao <= 10; iteracao++)
                // Como a thread é assíncrona, perdemos o vincúlo com a interface gráfica, logo, é necessário recuperarmos nossa
                    thread principal para fazer a atualização do UITextView
                // Note que quando a thread é síncrona, isso não é necessário
                // Abertura do processo paralelo assíncrono
                // A função dispatch_get_main_queue() retorna a thread principal da nossa aplicação
                dispatch_async(dispatch_get_main_queue(),^
                {
                    // Atualiza o UITextView com as informações atuais
                    [outputTabuadaTextView setText:[NSString stringWithFormat:@"%@\n%ld x %ld = %ld",[outputTabuadaTextView text],
                        (long)numeroAtual,(long)iteracao,(numeroAtual * iteracao)]];
                });
    });
}
```

```objc
-(IBAction)executarBlocoUnico:(UIButton *)sender
{
    // Variável de controle do bloco único
    static BOOL executado = NO;
    // Variável token para garantir a execução única do bloco seguinte
    // Similar a um "fosforo", só tem uso uma única vez, depois de queimado se torna inútil
    static dispatch_once_t tokenUnico;
    if(executado)
        // Atualiza o UILabel com as informações atuais
        [outputBlocoUnicoLabel setText:@"Bloco já fora executado!"];
    dispatch_once(&tokenUnico,^
    {
        executado = YES;
        // Atualiza o UILabel com as informações atuais
        [outputBlocoUnicoLabel setText:@"Executado bloco único!"];
    });
}
```

```objc
-(IBAction)baixarImagem:(UIButton *)sender
{
    [activityIndicator startAnimating];
    // Abertura de processo paralelo assíncrono
    // A constante "DISPATCH_QUEUE_PRIORITY_DEFAULT" representa o valor "0" e significa que irá retornar a thread mais bem avaliada e
        ociosa naquele momento de sua chamada
    // O segundo parametro "0" significa a não adição de flags para tal função, apenas o padrão
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,0),^
    {
        // Processo de download da imagem
        NSString * imagemCaminho = @"http://static2.wikia.nocookie.net/__cb20110204201432/bakuganrandomtalk/images/thumb/c/c8/
            TrollFace.png/768px-TrollFace.png";
        NSURL * imagemURL = [[NSURL alloc] initWithString:imagemCaminho];
        NSData * imagemData = [[NSData alloc] initWithContentsOfURL:imagemURL];
        UIImage * imagem = [[UIImage alloc] initWithData:imagemData];
        // Como a thread é assíncrona, perdemos o vincúlo com a interface gráfica, logo, é necessário recuperarmos nossa thread
            principal para fazer a atualização do UIImageView
        // Note que quando a thread é síncrona, isso não é necessário
        // Abertura do processo paralelo assíncrono
        // A função dispatch_get_main_queue() retorna a thread principal da nossa aplicação
        dispatch_async(dispatch_get_main_queue(),^
        {
            [activityIndicator stopAnimating];
            // Atualiza o UIImageView com a nova imagem
            [outputImagemImageView setImage:imagem];
        });
    });
}
```

```objc
-(IBAction)resetar:(UIButton *)sender
{
    [outputTabuadaTextView setText:nil];
    [outputBlocoUnicoLabel setText:nil];
    [outputImagemImageView setImage:nil];
}
```

- Finalizando, vamos implementar dois outros métodos

```objc
-(void)viewDidLoad
{
    [self iniciarInterface];
    [super viewDidLoad];

}
```

```objc
-(void)iniciarInterface
{
    [outputTabuadaTextView setText:nil];
    [outputTabuadaTextView setUserInteractionEnabled:YES];
    [outputTabuadaTextView setScrollEnabled:YES];
    [outputTabuadaTextView setEditable:NO];
    [outputBlocoUnicoLabel setText:nil];
    [activityIndicator setHidesWhenStopped:YES];
}
```

CONCLUINDO

- Neste tutorial, você aprendeu alguns conceitos importantes como:
- Tópicos de computação paralela
- Teoria e aplicação prática sobre GCD
- Tópicos sobre threads