Řízení běhu programu

Martin Hrubý

Seminář programování v iOS, FIT VUT v Brně

Motivace

- Program v Apple zařízení je typický:
  - Rozložením kódu programu do množství malých bloků.
  - Obtížně sledovatelným tokem událostí.
  - Asynchronním voláním funkcí.
  - Množstvím vláken. Protokoly všech pickerů.
- Zařízení Apple jsou typická:
  - Více-procesorovostí.
  - Podmínkami User Experience.

Koncepce více-vláknovosti

- Co je kód a co je vlákno. Vlákno a proces.
- NEočekává se kontinuální běh vlákna (NSThread). Vlákno ani nelze normálně suspendovat.
  - Běh vlákna zabírá výpočetní zdroje a bere energii.
  - Možné aplikace. Většinou lze převést na bloky (napojení na NSPort, NSRunLoop).
- Očekává se rozložení činnosti na elementy. Ty se staví do front.

Komunikace a řízení

- Zasílání zpráv, protokoly a delegates.
- Key-Value Observing (KVO).
- NSNotificationCenter a NSNotification.
- NSOperation. NSOperationQueue. NSThread.
- Grand Central Dispatch.
- XPC — více zavedeno na OS X, spekuluje se i o iOS.

Key-Value Observing

- Princip: objekt se registruje jako observer na klíč (property) jiného objektu.
  - observer musí implementovat NSKeyValueObserving proto.
  - pokud se změní hodnota key-value objektu, pak všichni observeři dostanou zprávu
  - jak se objekt dozví, že se jeho key-value změnilo?
- V Cocoa (OS X) dovedeno k dokonalosti.

Korektní nastavení hodnoty

```objc
@interface MJObject : NSObject

@property (nonatomic, strong) NSString *text;

@end

@implementation MJObject

-(void) setText:(NSString *)text
{
    [self willChangeValueForKey: @"text"];
    _text = text;
    [self didChangeValueForKey: @"text"];
}

@end
```

Sledování hodnoty

```
self.mjObject = [[MJObject alloc] init];
[self.mjObject addObserver: self
            forKeyPath: @"text"
               options: NSKeyValueObservingOptionNew |
NSKeyValueObservingOptionInitial
               context: nil];


-(void) observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
change:(NSDictionary *)change context:(void *)context
{
    NSLog(@"Key path change: %@, %@, %@", keyPath, object, change);
}

-(void) dealloc
{
    [self.mjObject removeObserver: self forKeyPath: @"text"];
}
```

Sleduje se keyPath

- [myObj addObserver:self forKeyPath:@"theArray.@count" options:0 context:NULL];
- + (NSSet *)keyPathsForValuesAffectingKey;
  - lze definovat observer na celý objekt.

NSNotificationCenter

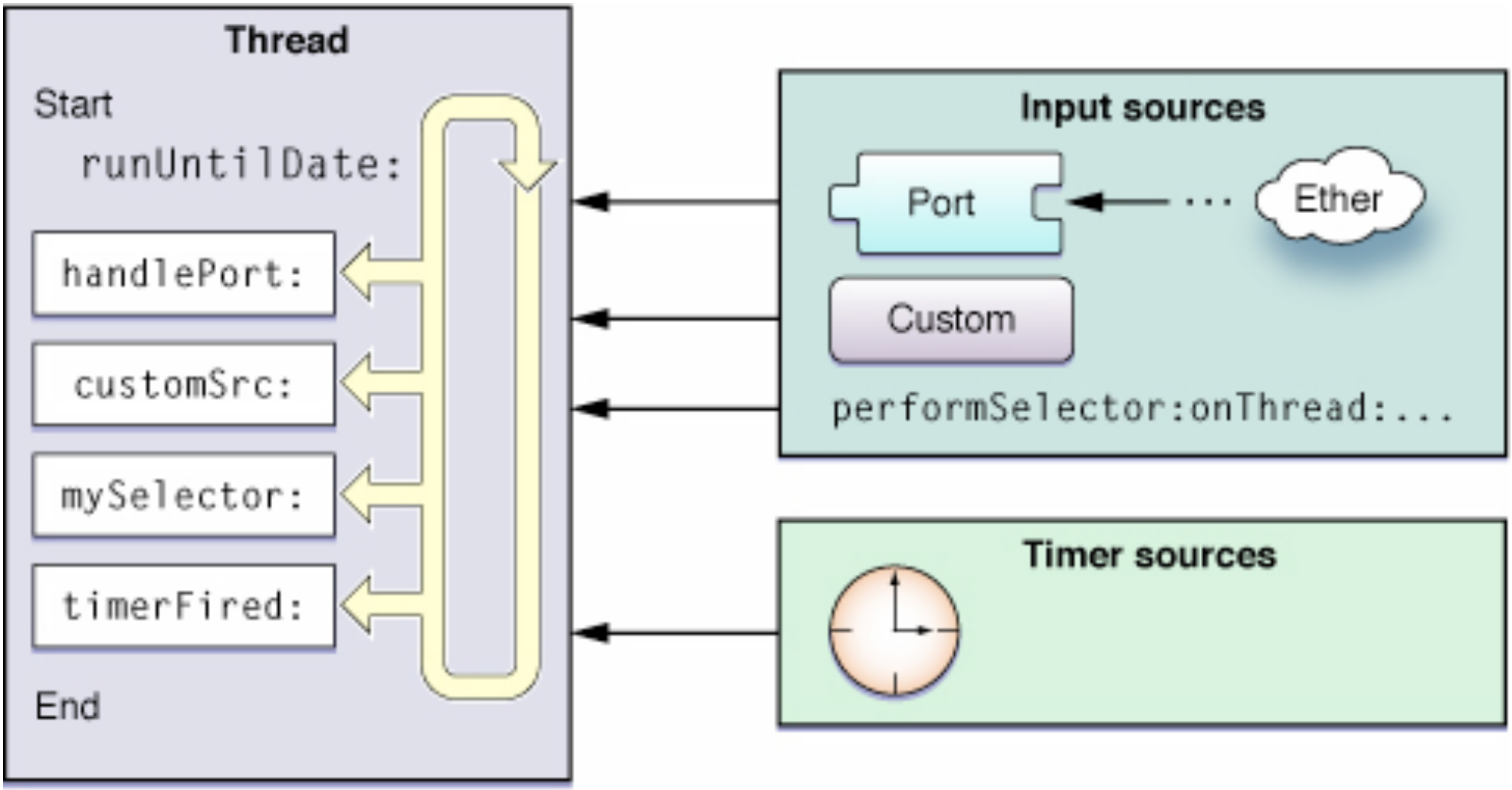
- Singleton pro aplikaci. Distributed varianta.
  - Registruje observery událostí. Srovnání s delegates.
- Výhodné pro události, kde je dynamický počet observerů.
- Při poslání zprávy do centra se okamžitě rozesílá do observerů, **tj ve stejném vlákně**.
  - Krátká reakce. Idealně GCD dispatch nějakého bloku.
  - Pozor při operacích s UI.

Notifikace demo

```
#define MOJEZPRAVA @"tohle-je-nazev"
-(void) obsluhaNotif: (NSNotification *) notif
{
}
-(void) dealloc
{
    NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];

    [nc removeObserver: self];
}
-(void) viewDidLoad {
    [super viewDidLoad];
    NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
    // registrace observeru
    [nc addObserver: self selector: @selector(obsluhaNotif:) name:
MOJEZPRAVA object: nil];
    // poslani zpravy
    NSNotification *notif = [NSNotification notificationWithName:
MOJEZPRAVA object: nil];
    [nc postNotification: notif];
    [nc postNotificationName: MOJEZPRAVA object: nil];
}
```

RunLoop


Thread
Start
runUntilDate:
handlePort:
customSrc:
mySelector:
timerFired:
End
Input sources
Port
Ether
Custom
performSelector:onThread:...
Timer sources

Bloky

- V C/C++: definujeme ukazatel na funkci a takto funkci předáme dál (a lze ji volat).
- Objective-C Block: předáme funkci a její provedení se zpracuje v datovém kontextu místa jejího vytvoření.
- Bloky jsou základ pro GCD.
- Asynchronní komunikace (“až to zjistíš, vykonej tento blok” versus “...pošli mi tuto zprávu”).

Příklad

```
__block BOOL found = NO;
NSSet *aSet = [NSSet setWithObjects: @"Alpha", @"Beta",
@"Gamma", @"X", nil];


NSString *string = @"gamma";

[aSet enumerateObjectsUsingBlock:^(id obj, BOOL *stop) {
    if ([obj localizedCaseInsensitiveCompare:string] == NSOrderedSame) {
            *stop = YES;
            found = YES;
        }
}];
```

I/O operace

```
[doc saveToURL:[doc fileURL]
      forSaveOperation:UIDocumentSaveForCreating
      completionHandler:^(BOOL success) {
        if (success) {
            [doc openWithCompletionHandler:^(BOOL success) {
                NSLog(@"new document opened from iCloud");
            }];
        }
    }];
```

NSTimer

- Jednorázové/opakované volání zvolené metody zvoleného objektu.
- Při přechodu aplikace do pozadí je nutno timery deaktivovat/smazat.
  - runLoop totiž neběží (výjimky),
  - po znovu-obnovení běhu app se všechny timery s prošlým časem vykonají najednou (chceme to tak?).
  - taky se mohou vykonat při suspendované aplikaci na pozadí

Timer příklad

```
-(void) timerReceiver:(NSNotification *)obj
{

}

-(void) timerActivate:(NSDate *)date
{
    self.myTimer = [[NSTimer alloc] initWithFireDate: date
                                                interval: 0
                                                  target: self
                                                selector:
@selector(timerActivate:)
                                                userInfo: nil
                                                 repeats:NO];
}
```

Fronty

- Kusy kódu jsou plánovany do front.
- RunLoop je z front vybírá a vlákna je vykonávají.
- Typy front:
  - sekvenční — jeden po druhém…
  - souběžné (concurrent)
- NSOperationQueue a GCD-fronty.

Systémové fronty

- Main Queue (sekvenční) — kód spuštený z této fronty smí přistupovat na UI.
  - změna obsahu UIView způsobí nastavení příznaku needsDisplay
  - pokud je nějaký UIView s tímto příznakem, pak se vyvolá překreslení UIWindow
- Global Queue (concurrent, 4 priority)
  - plánovat sem náročnější práci
  - pozor na thread-safety!!!

NSOperation

- GCD je založeno na blocích, avšak...
  - pokud potřebujeme složitější kontext tasku, je tu NSOperation
- NSOperation - abstraktní třída pro task.
- NSOperationQueue - fronta operací.
- NSOperation je **referencovaný objekt**, se kterým lze manipulovat:
  - priorita, precedenční podmínka, zrušení.

Kontrola nad plánovaným kódem

- Po vložení kódu do fronty ztrácíme kontrolu nad tím:
  - kdy bude spuštěn,
  - nelze jej vyřadit,
  - nelze jej zastavit/zabít.
- NSOperation lze poslat zprávu, bloku GCD nikoliv (je beztak příliš malý/elementární).
  - lze mu poslat uživatelskou zprávu. On ji musí chtít číst.

Uživatelská operace

```
@interface MyOperation :  NSOperation
@property (assign) double length;
@end

@implementation MyOperation

-(void) main
{
    NSLog(@"Operace %.2f spustena", self.length);
    [NSThread sleepForTimeInterval: self.length];
    NSLog(@"Operace %.2f dokoncena", self.length);
}

@end
```

Vnější kontext - fronta

```
MyOperation *opa = [[MyOperation alloc] init];
opa.length = 1.4;

NSOperationQueue *queue = [[NSOperationQueue alloc] init];
queue.maxConcurrentOperationCount = 10;

[queue addOperation: opa];
```

```
// "opa" zacne az po dokonceni opa2
[opa addDependency: opa2];

// akce provedena po ukonceni operace
[opa2 setCompletionBlock: ^{
     NSLog(@"Opa2 dokoncila");
}];
```

Grand Central Dispatch

- Kouskování kousků kódu k totálním atomům.
- Přehazování toku programu mezi vlákny.
- Typicky:
  - práce se dělá v Global Queue
  - UI se dělá v Main Queue

Dispatching

- Synchronní:
  - blok vloží/naplánuje sub-blok do fronty
  - časem se sub-blok dokončí
  - pak se pokračuje dále blokem
- Asynchronní:
  - blok vloží/naplánuje sub-blok do fronty
  - … a dále bez přerušení pokračuje

Demo

```
void    simple_gcd_demo()
{
    dispatch_queue_t myCustomQueue;
    myCustomQueue = dispatch_queue_create("mojeFronta", NULL);

    dispatch_async(myCustomQueue, ^{
        printf("Do some work here.\n");
    });

    printf("The first block may or may not have run.\n");

    dispatch_sync(myCustomQueue, ^{
        printf("Do some more work here.\n");
    });

    printf("Both blocks have completed.\n");
}
```

Demo: rozděl práci

```
void    prace(id obj)
{
    NSNumber *arg = (NSNumber *) obj;

    [NSThread sleepForTimeInterval: [arg doubleValue]];
    NSLog(@"Prace %@ dokoncena", arg);
}
```

Vše synchronně

```
void    rozdel_praci(NSArray *tasks)
{
    for (NSNumber *pr in tasks) {

   dispatch_sync(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAU
   LT, 0), ^{
            prace(pr);
        });
    }
}
```

```
 int main() {
  NSLog(@”Prace zacina”);
   rozdel_praci(@[@(7),@(2.1),@(0.2),@(3)]);
  NSLog(@”Prace dokoncena”);
 }
```

```
2013-03-27 11:12:14.151 gcd[16135:303] Prace zacina
2013-03-27 11:12:21.153 gcd[16135:303] Prace 7 dokoncena
2013-03-27 11:12:23.256 gcd[16135:303] Prace 2.1 dokoncena
2013-03-27 11:12:23.459 gcd[16135:303] Prace 0.2 dokoncena
2013-03-27 11:12:26.461 gcd[16135:303] Prace 3 dokoncena
2013-03-27 11:12:26.462 gcd[16135:303] Prace hotovo
```

... dispatch async

```
2013-03-27 11:13:35.159 gcd[16147:303] Prace zacina
2013-03-27 11:13:35.162 gcd[16147:303] Prace hotovo
```

... dispatch async, caller čeká

```
NSLog(@"Prace zacina");
 rozdel_praci(@[@(7),@(2.1),@(0.2),@(3)]);
 [NSThread sleepForTimeInterval: 20];
NSLog(@"Prace hotovo");
```

```
2013-03-27 11:14:43.457 gcd[16157:303] Prace zacina
2013-03-27 11:14:43.663 gcd[16157:1c03] Prace 0.2 dokoncena
2013-03-27 11:14:45.561 gcd[16157:1b03] Prace 2.1 dokoncena
2013-03-27 11:14:46.461 gcd[16157:2403] Prace 3 dokoncena
2013-03-27 11:14:50.461 gcd[16157:1403] Prace 7 dokoncena
2013-03-27 11:15:03.461 gcd[16157:303] Prace hotovo
```

```
// fronta je seriova
dispatch_queue_t q = dispatch_queue_create("mojeFronta", NULL);

    for (id pr in tasks) {
        NSLog(@"Zarazuju task %@", pr);
        dispatch_async(q, ^{
            prace(pr);
        });
    }

  // Cekani na dokonceni tasku ve fronte (vyprazdneni fr.)
    dispatch_sync(q, ^{
        NSLog(@"Fronta dokoncena");
    });
```

```
2013-03-27 11:32:24.536 gcd[16335:303] Prace zacina
2013-03-27 11:32:24.538 gcd[16335:303] Zarazuju task 7
2013-03-27 11:32:24.539 gcd[16335:303] Zarazuju task 2.1
2013-03-27 11:32:24.539 gcd[16335:303] Zarazuju task 0.2
2013-03-27 11:32:24.540 gcd[16335:303] Zarazuju task 3
2013-03-27 11:32:31.540 gcd[16335:1b03] Prace 7 dokoncena
2013-03-27 11:32:33.643 gcd[16335:1b03] Prace 2.1 dokoncena
2013-03-27 11:32:33.845 gcd[16335:1b03] Prace 0.2 dokoncena
2013-03-27 11:32:36.847 gcd[16335:1b03] Prace 3 dokoncena
2013-03-27 11:32:36.849 gcd[16335:303] Fronta dokoncena
2013-03-27 11:32:36.849 gcd[16335:303] Prace hotovo
```

Bezpečné řešení paralelně

```
void    rozdel_praci(NSArray *tasks)
{
    dispatch_queue_t q =
    dispatch_queue_create("mojeFronta", DISPATCH_QUEUE_CONCURRENT);
    dispatch_group_t group = dispatch_group_create();

    for (id pr in tasks) {
        NSLog(@"Zarazuju task %@", pr);

        dispatch_group_async(group, q, ^{
            prace(pr);
        });
    }

    dispatch_group_wait(group, DISPATCH_TIME_FOREVER);
}
```

Paralelní operace nad UI

- Výpočetní operace do global_queue
- Kreslící operace do main_queue
- Všechny -()view… jsou v Main Queue

```
// Potřebujeme vyhodnotit nějakou časově náročnou operaci
    dispatch_async(dispatch_get_global_queue(0, 0), ^{

        // Výpočet
        // ...

        // zařazení vykreslení výsledků do UI
        dispatch_sync(dispatch_get_main_queue(), ^{
            // ...
        });
    });
```

Natahování obrázků do UITableViewCell

- Každá buňka má atribut “imageView”.
- Seznam má reprezentovat jednotlivé obrázky v adresáři.

Synchronní řešení

```
-(UITableViewCell *) tableView: (UITableView *)tableView
         cellForRowAtIndexPath: (NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView
                             dequeueReusableCellWithIdentifier: @"cell"]

    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithStyle:
UITableViewCellStyleSubtitle reuseIdentifier: @"cell"];
    }

    Course *course = [data objectAtIndex: indexPath.row];
    cell.textLabel.text = course.name;

    // Ted chci dodat obrazek
    cell.imageView.image = [UIImage imageWithData:
                            [self imageDataForIndexPath:indexPath]];

    return cell;
}
```

Asynchronní řešení

```
// Ted chci dodat obrazek (mam defaultni obr)
    cell.imageView.image = self.picturePlaceholder;

    dispatch_async(dispatch_get_global_queue(0, 0), ^{
        NSData *obrData = [self imageDataForIndexPath: indexPath];
        UIImage *image = [UIImage imageWithData: obrData];

        dispatch_sync(dispatch_get_main_queue(), ^{
            // pozor !!! cell uz nemusi byt platne
            UITableViewCell *pcell =
         [tableView cellForRowAtIndexPath:indexPath];
            // pcell muze byt nil - nevadi
            pcell.imageView.image = image;
            [pcell setNeedsLayout];
        })
    });
```

NSOperation versus GCD

- GCD je velmi komfortní
  - množství malých bloků, vyladěná efektivita
- NSOperation
  - máme stále možnost referencovat jednotlivé činnosti
  - můžeme specifikovat precedence

Proč si komplikujeme program

- Program může existovat bez znalosti GCD.
  - Celý poběží v Main Queue.
  - Bude tím trpět UI, User Experience.
  - Výpočetní práce se musí vždy nějak udělat. Rychlost aplikace je subjektivní pozorování uživatele, tj. je to rychlost UI.
- Vkročením do Global Queue se otvírá Pandořina skříňka.
- Lze se tomu vyhnout?

Použití

- AppDidFinishLaunching:
  - náročnější inicializaci přesunout ASYNC do Global Queue
- Příjem notifikace
  - obsloužit co nejrychleji, ASYNC do Main/Global
- Události se zápisem do UI
  - ASYNC do Main Queue

Závěr

- Rychlost aplikace má subjektivní a objektivní podobu:
  - odezva UI
  - efektivnost algoritmu výpočtu
- NSNotifCenter — koncepční rozesílání zpráv.
- KVO — koncepční rozesílání (synchronizace) hodnoty klíčů.