**Розробка під macOS**

**1 Огляд системи, особливості**

macOS - це POSIX-сумісна операційна система Darwin, яка є вільним програмним забезпеченням. Його ядром є XNU, який використовує мікроядро Mach і стандартні сервіси BSD. Всі функції Unix в macOS доступні через консоль.

**1.1 Огляд файлових систем macOS**

Комп'ютери Apple використовують два основних типи файлових систем: HFS+ і APFS.

**HFS+** використовує деревоподібну структуру, яка називається B\*-деревом, для зберігання більшості метаданих.   
**APFS** збільшує швидкість читання/запису даних з SSD, а також збільшує простір для зберігання, підраховуючи вільний дисковий простір.

Особливості APFS:

* Повнодискове шифрування;
* 64-бітна архітектура;
* Розподіл простору - якщо в одному умовному розділі недостатньо місця для установки файлу, простір іншого буде використано автоматично;
* Знімки - використовуються для більш ефективного резервного копіювання;
* Клони - практично миттєве копіювання файлу або каталогу, яке не вимагає додаткового місця для зберігання. При зміні клону файлова система фіксує тільки зміну даних. Таким чином, нова файлова система може зберігати безліч версій великих файлів, займаючи менше місця на диску.
  1. **Огляд безпеки macOS**

**Keychain** - це зашифрований контейнер, який надійно зберігає паролі, облікові записи та іншу конфіденційну інформацію, яка зазвичай використовується в повсякденному житті.

**Захист цілісності системи(SIP)** - це функція, яку Apple розробила для забезпечення високого рівня безпеки системи.

SIP включає в себе наступні механізми безпеки:

- захист файлової системи та дозволів

- захист процесів від введення коду

- Захист від встановлення непідписаних розширень ядра.

**Code signing** – використовується для підтвердження того, що додаток створено конкретним розробником. Після підписання заявки система може виявити будь-яку зміну в додатку, незалежно від того, чи була зміна зроблена випадково або шкідливим кодом.

**Нотаріальне засвідчення** - це процес перевірки підписаної Apple заявки на наявність шкідливих компонентів.

**1.3 Пакети додатків**

**Пакет** - це файловий каталог з певною структурою та розширенням файлу, що дозволяє групувати пов'язані файли як концептуально єдиний елемент.

Пакет потрібен для спрощення процесу встановлення та видалення програми, тому для встановлення програми користувачеві просто потрібно скопіювати пакет на комп'ютер, а щоб видалити його, просто видалити цей пакет.

Додаток application bundle для macOS:

|  |
| --- |
| MyApp.app/  Context/  Info.plist  MacOS/  Resources/ |

|  |  |
| --- | --- |
| **Каталог/файл** | **Опис:** |
| MacOS | (Обов'язково) Містить виконуваний файл програми, який запускається при подвійному клацанні на зв'язці. |
| Resources | Містить всі ресурсні файли програми. |
| Frameworks | Містить будь-які фреймворки,бібліотеки бібліотек, які використовуються виконуваним файлом |
| Info.plist | Містить інформацію для ідентифікації програми. Файл містить дані у форматі xml, що визначає конфігурацію цього пакета. |

* 1. **Огляд виконуваного файлу Macho**

**Файл Macho** є власним форматом виконуваного/двійкового файлу в macOS. Двійковий формат визначає порядок зчитування даних і коду в пам'ять системи.

Файл Macho складається з трьох частин:

**1 Header** - саме тут зберігається загальна інформація про файл: магічне число (бітність архітектури), архітектура, тип файлу (liba, driver, bundle і т. Д.), Кількість завантажуваних команд і т. Д.

**2 Load commands** - основна частина файлу Mach-O. Існує безліч різних команд завантаження, розбір яких залежить від типу файлу, зазначеного в заголовку.

Завантажувальні команди підкажуть, як і куди завантажувати частини Mach-O: сегменти, таблиці символів і від яких бібліотек залежить файл, щоб завантажити їх першим.

Деякі команди завантаження:

* LC\_SEGMENT - містить різну інформацію про певний сегмент: розмір, кількість секцій, зміщення у файлі і, після завантаження, в пам'яті;
* LC\_SYMTAB — завантажує таблицю символів і рядків;
* LC\_DYSYMTAB — створює таблицю імпорту;
* LC\_LOAD\_DYLIB — вказує залежність від деякої сторонньої бібліотеки

**3 Segments** - описують області пам'яті, куди потрібно завантажувати розділи з кодом або даними.

Найбільш важливими сегментами є наступні:

* \_\_TEXT — виконуваний код та інші дані тільки для читання;
* \_\_DATA - дані, доступні для запису; включаючи таблиці імпорту, які, як правило, змінюються динамічним завантажувачем при пізньому зв'язуванні;
* \_\_OBJC - різноманітна інформація стандартної бібліотеки середовища виконання мови Objective-C;
* \_\_LINKEDIT - тут динамічний завантажувач розміщує свої дані для вже завантажених модулів: таблиці символів, рядки і т.д.

Для перегляду вмісту файлу macho ви можете скористатися або консольною утилітою otool, або утилітою MachoView.

**2. Objective C**

**2.1 Огляд мови**

Objective C — об'єктно-орієнтована мова програмування, сумісна з C/C++. Objective C орієнтована на повідомлення, тобто виклики методів інтерпретуються не як виклик функції, а як відправка повідомлення (з ім'ям і аргументами) об'єкту.

## **2.2 Синтаксис**

Клас описується двома файлами: header (. (h) та реалізація (. м).

Приклад:

|  |
| --- |
| //  // Book.h  //  #import <Foundation/Foundation.h>  NS\_ASSUME\_NONNULL\_BEGIN  @interface Book : NSObject  {  int \_pageCount;  }  @property (nonatomic, readonly) NSString\* name;  - (nonnull NSString\*)showInfo;  @end  NS\_ASSUME\_NONNULL\_END  //  // Book.m  //  #import "Book.h"  @implementation Book  - (instancetype)initWithName:(nonnull NSString\*)name pageCount:(int)pageCount  {  self = [super init];  if (self)  {  \_name = name;  \_pageCount = pageCount;  }  return self;  }  - (nonnull NSString\*)showInfo  {  return [NSString stringWithFormat:@"Name: %@. Page count: %d", \_name, \_pageCount];  }  @end |

**2.3 Властивості**

Властивість надає методи setter і getter при доступі до даних.

приклад:

|  |
| --- |
| @property (неатомарне, лише для читання) ім'я NSString\*; |

У властивість можуть мати різні типи:

* атрибути доступності: readwrite / readonly;

##### nullability атрибут: nonnull / nullable;

##### атрибут атомарности: атомарні / неатомні;

##### атрибуты володіння: retain / strong / copy / weak;

**2.4 Категорії**

Категорії використовуються для розширення функціональності існуючого класу.

приклад:

|  |
| --- |
| @interface NSString (MyAdditions)  +(NSString \*)getCopyRightString;  @end  @ implementation NSString(MyAdditions)  +(NSString \*)getCopyRightString {  return @"Copyright macOSDevelopment.com 2021";  }  @end |

**2.5 Протоколи**

Протоколи дозволяють визначити методи, які, як очікується, будуть реалізовані в класах, які успадковують певний протокол. Протоколи дуже схожі на інтерфейси на C#.

Приклад:

|  |
| --- |
| @protocol TestProtocol  @required  -(void) test;  @optional  -(void) testWithName: (NSString \*) name;  @end  @interface TestClass : NSObject <TestProtocol, NSCoding>  ...  @end |

**2.6 Управління пам'яттю**

Існує два методиуправління пам'яттю в Objective-C:

* "Ручне утримання-вивільнення" або MRR
* "Автоматичний підрахунок посилань" або ARC

Ручне утримання-вивільнення – явне управління пам'яттю. Метод заснований на кількості посилань, які надає NSObject.

приклад:

|  |
| --- |
| @interface SampleClass:NSObject  - (void)sampleMethod;  @end  @implementation SampleClass  - (void)sampleMethod {     NSLog(@"Hello, World! \n");  }  - (void)dealloc  {    NSLog(@"Object deallocated");    [super dealloc];  }  @end    int main() {       SampleClass \*sampleClass = [[SampleClass alloc]init];     [sampleClass sampleMethod];   NSLog(@"Retain Count after initial allocation: %d",     [sampleClass retainCount]);     [sampleClass retain];       NSLog(@"Retain Count after retain: %d", [sampleClass retainCount]);     [sampleClass release];     NSLog(@"Retain Count after release: %d", [sampleClass retainCount]);     [sampleClass release];     NSLog(@"SampleClass dealloc will be called before this");       sampleClass = nil;     return 0;  } |

Вивід:

2013-09-28 04:39:52.310 demo[8385] Hello, World!

2013-09-28 04:39:52.311 demo[8385] Retain Count after initial allocation: 1

2013-09-28 04:39:52.311 demo[8385] Retain Count after retain: 2

2013-09-28 04:39:52.311 demo[8385] Retain Count after release: 1

2013-09-28 04:39:52.311 demo[8385] Object deallocated

2013-09-28 04:39:52.311 demo[8385] SampleClass dealloc will be called before this

Автоматичний підрахунок посилань - це підхід до управління пам'яттю, тільки тут підрахунок посилань на об'єкт бере на себе сама система.

приклад:

|  |
| --- |
| @interface SampleClass:NSObject  - (void)sampleMethod;  @end  @implementation SampleClass  - (void)sampleMethod {     NSLog(@"Hello, World! \n");  }  - (void)dealloc  {    NSLog(@"Object deallocated");  }  @end  int main() {     @autoreleasepool {        SampleClass \*sampleClass = [[SampleClass alloc]init];        [sampleClass sampleMethod];        sampleClass = nil;     }     return 0;  } |

Вивід:

2013-09-28 04:45:47.310 demo[8385] Hello, World!

2013-09-28 04:45:47.311 demo[8385] Object deallocated

**3. Swift**

Swift - це нова мова програмування, розроблена Apple для розробки iOS та macOS. Swift поєднує в собі найкраще з C і Objective-C, без обмежень сумісності з C.

**3.1 Синтаксис**

Оголошення змінних:

|  |
| --- |
| var variableName = <початкове значення> |

Ви можете додати таку анотацію:

|  |
| --- |
| var variableName:<тип даних> = <необов'язкове початкове значення> |

Тип може бути необов'язковим, тобто змінна може містити значення вказаного типу і взагалі нічого:

|  |
| --- |
| var perhapsInt: Int? |

Оголошення констант:

|  |
| --- |
| let variableName = <initial value> |

Синтаксис функції:

|  |
| --- |
| func funcname(Parameters) -> returntype {     Statement1     Statement2     ---     Statement N     return parameters  } |

Синтаксис класу:

|  |
| --- |
| //  //  SwiftBook.swift  //  SwiftBook  //  import Foundation  class SwiftBook  {      private let \_pageCount: Int;      private(set) var \_name: String;      init(name: String, pageCount: Int) {          \_pageCount = pageCount;          \_name = name;      }      func showInfo() -> String {          return “Name: \( name). Page count: \( pageCount)”      }  }  //  //  main.swift  //  SwiftBook  //  import Foundation  var book: SwiftBook = SwiftBook(name: "New Book", pageCount: 100);  print(book.showInfo()) |

**3.2 Закриття**

Закриття - це автономні блоки з певною функціональністю, які можна передавати і використовувати у вашому коді. Замикання в Swift аналогічні блокам в C і Objective-C, а лямбди в інших мовах програмування.

Приклад:

|  |
| --- |
| let divide = {     (val1: Int, val2: Int) -> Int in     return val1 / val2  }  let result = divide(200, 20)  print (result) |

**3.3 Дженерики**

Дженерики використовуються для написання гнучких і багаторазових функцій і типів.

Приклад:

|  |
| --- |
| func exchange<T>(a: inout T, b: inout T) {     let temp = a     a = b     b = temp  }  var numb1 = 100  var numb2 = 200  print("Before Swapping Int values are: \(numb1) and \(numb2)")  exchange(a: &numb1, b: &numb2)  print("After Swapping Int values are: \(numb1) and \(numb2)")  var str1 = "Generics"  var str2 = "Functions"  print("Before Swapping String values are: \(str1) and \(str2)")  exchange(a: &str1, b: &str2)  print("After Swapping String values are: \(str1) and \(str2)") |

**3.4 Розширення**

Розширення такі ж, як категорії в Objective C. Дозволяють розширити функціональність існуючого класу.

Приклад:

|  |
| --- |
| extension Int {     var add: Int {return self + 100 }     var sub: Int { return self - 10 }     var mul: Int { return self \* 10 }     var div: Int { return self / 5 }  }  let addition = 3.add  print("Addition is \(addition)")  let subtraction = 120.sub  print("Subtraction is \(subtraction)")  let multiplication = 39.mul  print("Multiplication is \(multiplication)")  let division = 55.div  print("Division is \(division)")  let mix = 30.add + 34.sub  print("Mixed Type is \(mix)") |

**4. Оbjective C проти Swift**

Порівняння Objective C и Swift:

1. **Продуктивність**  
   Swift забезпечує високу швидкість виконання програм і працює в 2,6 рази швидше, ніж Objective C. Основною причиною того, що Objective C працює повільніше, є використання компіляції в коді середовища виконання. Це означає, що коли об'єкт Objective C викликає інший об'єкт в коді, виникає додатковий рівень неспрямованого доступу, який є вартістю виклику декількох доступів.
2. **Безпека**

Objective C надає можливість викликати метод з нульовим покажчиком. У такий момент нічого не відбувається. Такий вираз в коді називається no-operation (NOP). На перший погляд це здається корисним, так як збоїв в разі звернення до нульового покажчика не буде, але призводить до непередбачуваних результатів, які складно усунути і важко знайти.

Swift негайно видає помилку, якщо доступ до нульового вказівника.

1. **Підтримка**

Розробники Objective-C підтримують два окремих кодових файли для підвищення ефективності та часу розробки програми.

Swift підтримує один файл програмного коду.

1. **Управління пам'яттю**

Objective C використовує ARC для управління пам'яттю об'єктно-орієнтованогокоду. Однак для C (наприклад, Core Graphics) ARC не підтримується.

Swift також використовує ARC, якапідтримується для всіх API.

**5. Фреймворки**

Фреймворк - це ієрархічний каталог (зв'язка), який інкапсулює загальні ресурси: динамічні бібліотеки, файли nib, зображення, файли локалізації рядків, заголовки, документацію в одному пакеті.

Популярні фреймворки для розробки macOS:

* Foundation є базовим фреймворком, який надає класи примітивних об'єктів: NSString, NSMutableString, NSArray, NSDictionary, NSSet, Notifications і т.д.
* CoreData – містить інтерфейси для управління моделями даних (кеш, база даних).
* CFNetwork – містить інтерфейси для взаємодії з мережею за допомогою HTTP, сокетів тощо.
* AVFoundation – надає інтерфейси для відтворення, запису, редагування аудіо файлів.
* CoreGraphics – надає інтерфейси для роботи з графічним контентом.
* IOKit – містить основні інтерфейси для створення драйверів, взаємодіючих з драйверами ядра.

**Література:**

1. Внутрішні компоненти macOS:

* <https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/OSX_Technology_Overview/About/About.html#//apple_ref/doc/uid/TP40001067>
* http://newosxbook.com/MOXiI.pdf

1. Файлова система:

* <https://ru.bmstu.wiki/HFS_Plus>
* <https://habr.com/ru/post/395253/>
* <https://ddr5.com.ua/faylovye-sistemy-mac-os/>

1. Комплекти додатків:

* <https://developer.apple.com/library/archive/documentation/CoreFoundation/Conceptual/CFBundles/BundleTypes/BundleTypes.html#//apple_ref/doc/uid/10000123i-CH101-SW1>

1. Безпека:

* <https://developer.apple.com/documentation/security/notarizing_macos_software_before_distribution>
* <https://developer.apple.com/library/archive/documentation/Security/Conceptual/CodeSigningGuide/Introduction/Introduction.html>

1. Mthink files:

* <https://redmaple.tech/blogs/macho-files/>
* <https://developer.apple.com/library/archive/documentation/Performance/Conceptual/CodeFootprint/Articles/MachOOverview.html>

1. Завдання С:

* <https://www.tutorialspoint.com/objective_c/index.htm>
* <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html>

1. Swift:

* <https://docs.swift.org/swift-book/GuidedTour/GuidedTour.html>
* <https://www.tutorialspoint.com/swift/swift_closures.htm>
* https://swiftbook.ru/content/swift-tour/

1. Фреймворки macOS:

* https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/OSX\_Technology\_Overview/SystemFrameworks/SystemFrameworks.html