Тема 4. Архітектура мобільних платформ. Характеристики платформ для мобільних пристроїв

4.1 Архітектура платформи Android

За широтою можливостей платформа Android не поступається операційним системам настільних ПК. Це багаторівневе середовище на основі ядра Linux з великими функціональними можливостями. У підсистему інтерфейсу, призначеного для користувача, входять:

- вікна;
- подання;
- віджети для відображення загальних елементів, таких, як поля, що редагуються, списки та списки, що розгортаються.

В Android вбудовано браузер з движком WebKit з відкритим вихідним кодом, який лежить в основі браузера Safari мобільного телефону iPhone.

Android має широкий спектр можливостей для підключення, бездротового зв'язку з використанням Wi-Fi, Bluetooth та протоколів передачі даних через стільникову мережу (GPRS, EDGE, 3G і ін.). В Android активно використовуються сервіси, які надаються Google; наприклад, в своїх прикладних програмах можна посилатися на Google Maps для позиціонування пристрою. В стек програмного забезпечення Android входить також підтримка сервісів, заснованих на визначенні місця розташування (наприклад, GPS), та акселерометрів, хоча не усі пристрої на цій платформі оснащені необхідним обладнанням. Присутня також підтримка відеокамери.

Через обмеження ресурсів мобільні пристрої завжди мали великі проблеми з обробкою графіки/мультимедіа та способами зберігання даних, на відміну від персональних комп'ютерів. Для вирішення даної проблеми платформа Android пропонує вбудовану підтримку 2-D і 3-D графіки, з використанням бібліотеки OpenGL. Для забезпечення зберігання даних платформа Android використовує популярну базу даних з відкритим вихідним кодом SQLite. На рис. 4.1 зображена спрощена схема рівнів програмного забезпечення Android.

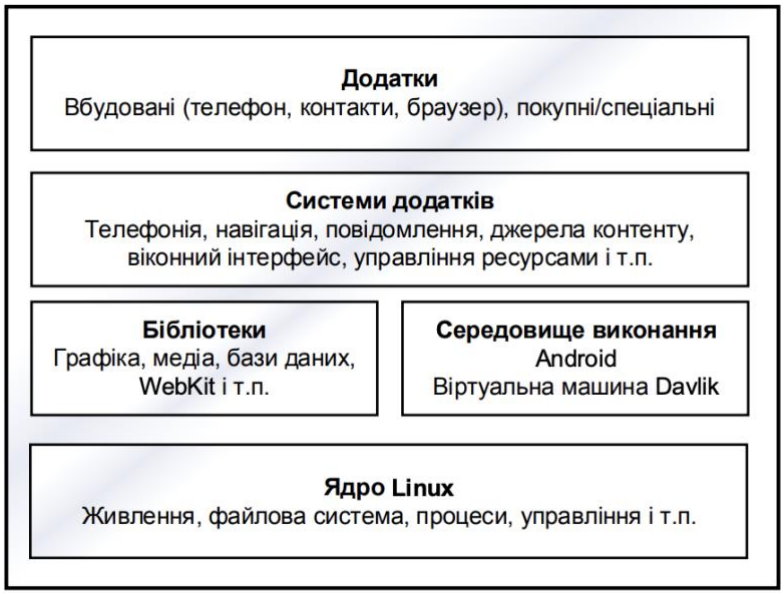

Рисунок 4.1 – Рівні програмного забезпечення Android

4.1.1 Віртуальна машина Dalvik

Операційна система Android працює поверх ядра Linux. Для створення Android-додатків спочатку використовувалася мова програмування Java, а потім виконувалися прикладні програми у віртуальній машині (VM). Необхідно звернути увагу на те, що віртуальна машина – це не віртуальна машина Java (JVM), а відкрита технологія Dalvik Virtual Machine. При запуску програми Android створюється і запускається окремий екземпляр Dalvik VM, який, в свою чергу, розташований в межах керованого ядром Linux процесу, як показано на рис. 4.2.

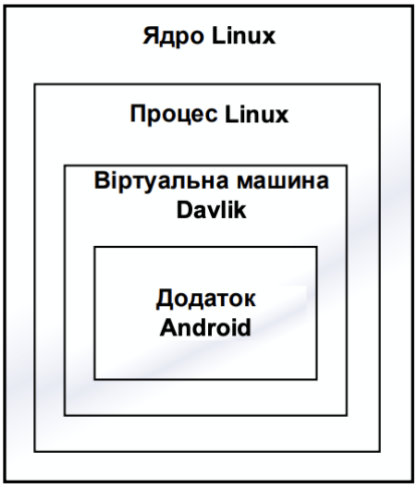

Рисунок 4.2 – Віртуальна машина Dalvik

Dalvik є віртуальною машиною в межах операційної системи Android від Google, яка виконує прикладні програми, що написані для Android. Dalvik є невід'ємною частиною стека програмного забезпечення Android в ОС Android версії 4.4 «KitKat» та більш ранніх версій, які зазвичай використовуються в мобільних пристроях, таких, як мобільні телефони та планшетні комп'ютери, а останнім часом на таких пристроях, як смарт-телевізори. Dalvik є програмним забезпеченням з відкритим вихідним кодом, написаним Dan Bornstein, який назвав його на честь рибальського селища Dalvik в Ісландії.

Програми для Android, як правило, написані на Java та скомпільовані в байт-код для віртуальної машини Java, яка потім транслюється на Dalvik байткод і зберігається в .dex (Dalvik Executable) і .odex (оптимізований Dalvik Executable) файлах. Компактний формат Dalvik Executable призначений для систем, які обмежені з точки зору пам'яті та швидкості процесора.

Правонаступником Dalvik є Android Runtime (ART), який використовує той самий байт-код та .dex файли (але не .odex файли), з послідовністю, спрямованою на підвищення продуктивності для кінцевих користувачів.

4.1.2 Віртуальна машина ART

Android Runtime (ART) є середовищем виконання прикладних програм, яке використовується Android. ART виконує перетворення байт-коду програми в інструкції, які потім виконуються за допомогою середовища виконання пристрою.

В Android 2.2 «Froyo» з'явилися JIT-компіляція в Dalvik, оптимізація виконання прикладних програм за допомогою постійно профілюючих програм.

У той час як Dalvik інтерпретує частину в байт-коді прикладної програми, виконання ART цих коротких відрізків байт-коду, яке називається «слідом», забезпечує значне підвищення продуктивності.

На відміну від Dalvik, ART вводить використання ahead-of-time (AOT) компіляцію шляхом компіляції цілих прикладних програм в машинний код при їх установці. Усуваючи інтерпретації Dalvik на основі JIT-компіляції, ART

підвищує загальну ефективність виконання та знижує енергоспоживання, що приводить до підвищення автономії батареї на мобільних пристроях. Також ART забезпечує більш швидке виконання програм, поліпшений розподіл пам’яті та механізми збирання сміття (GC), нові можливості налагодження прикладних програм, а також більш точне профілювання прикладних програм високого рівня.

Для забезпечення зворотної сумісності ART використовує такий самий вхідний байт-код, що і Dalvik, який надається через стандартні .dex файли як частина APK файлів, в той час як .odex файли замінюються Executable and Linkable Format (ELF) файлами.

Після того як прикладна програма буде зібрана та скомпільована на пристрої, утиліта dex2oat запускається з виконуваного ELF-файлу. У результаті ART усуває відмінності виконання прикладних програм, накладні витрати, пов’язані з інтерпретацією Dalvik, та трасування на основі JIT компіляції.

ART вимагає додаткового часу для компіляції при установці прикладної програми, а прикладні програми вимагають трохи більше місця для зберігання (як правило використовується флеш-пам’ять) скомпільованого коду.

В Android 4.4 «KitKat» з’явилася технологія попереднього огляду ART, в тому числі як альтернативного середовища виконання та збереження, замість Dalvik, як віртуальної машини поза вибором. У подальшому в Android 5.0 «Lollipop» Dalvik був повністю замінений на ART.

4.1.3 Керування ресурсами мобільних пристроїв

Ядро, що використовується в Android, є модернізованим ядром серії Linux для забезпечення виконання деяких особливих потреб мобільних платформ.

Функції ядра в основному поширюються на драйвери, керування живленням, засоби керування та коригування обмежених можливостей Android. Функції керування живленням мають вирішальне значення для мобільних пристроїв.

Ядро Android знаходиться у вільному доступі. Всі зміни можна відстежувати через загальнодоступне сховище Android. Є кілька ядер, доступних в репозиторії. Деякі апаратні специфічні ядра для платформ, таких, як MSM7xxx, Open Multimedia Application Platform (OMAP) та Tegra, також доступні в сховищі.

Зміни, які вносяться до базової версії ядра, можна розділити на такі: виправлення помилок, засоби для підвищення простору для користувача (low memory killer, binder, ash mem, logger і т. д.), нова інфраструктура (особливо wake locks), підтримка нових SoCs (msm7k, msm8k і т. д.) та плат/пристроїв. В майбутньому специфічні ядра Android та базові ядра Linux повинні об'єднатися, але цей процес йде повільно і забере деякий час.

Android виділяє кожній прикладній програмі, яка призначена для користувача, окремий адресний простір. Програми, що працюють в Dalvik VM або ART, періодично переходять в сплячий режим або режим очікування. Це важливо, тому що поза вибором Android намагається перевести систему в режим сну або в режим очікування якнайшвидше.

При цьому екран залишається включеним, або CPU не спить, щоб швидше реагувати на пробудження. Інструменти Android, які використовуються для вирішення цього завдання, називаються блокіраторами засипання (wakelocks).

Функції wakelock можуть бути отримані компонентами ядра або адресним простором процесу користувача.

Інтерфейс користувача для створення блокіраторів використовує файл /sys/power/wakelock, в якому записується ім'я нового блокіратора. Для зняття блокування процес записує ім'я файлу в /sys/power/wakeunlock. Для блокування можна вказати час, після якого воно спрацює автоматично. Всі системи, які використовують на даний час сервіс блокування, вказуються у файлі /proc/wakelocks.

Інтерфейс ядра для блокіраторів пробудження дозволяє вказати, чи повинен wakelock запобігати низькому енергоспоживанню чи припиненню роботи системи. WakeLock створюється процесом wakelockinit() та вилучається

wakelockdestroy(). Створений блокіратор може бути запущений процесом wakelock() та розблокований wakeunlock(). Як і в просторі користувача, можна визначити тайм-аут для блокування. Концепція блокіраторів глибоко інтегрована в Android у вигляді драйверів; багато прикладних програм інтенсивно використовують їх.

Клас PowerManager – це службовий клас, який дозволяє віртуальним машинам Андроїд отримати доступ до можливостей WakeLock драйвера керування живленням. PowerManager забезпечує чотири різних види блокування:

- PARTIAL WAKE LOCK – CPU не спить, навіть якщо кнопка живлення пристрою натиснута.
- SCREEN DIM WAKE LOCK – блокування екрана залишається увімкненим, але екран залишається сірим.
- SCREEN BRIGHT WAKE LOCK – екран блокується з нормальною яскравістю.
- FULL WAKE LOCK – блокування клавіатури та екрана зі звичайним підсвічуванням.

Тільки PARTIAL WAKE LOCK гарантує, що процесор повністю увімкнутий, решта три типи дозволяють процесору засипати після того, як кнопка живлення пристрою натиснута. Як і блокування адресного простору прикладних програм, блокування, що надаються PowerManager, можуть бути об’єднані з тайм-аутом. Крім того, можна розбудити пристрій при включенні екрана.

Керування пам’яттю пов’язане зі змінами у базовому ядрі для поліпшення використання пам’яті в системах з невеликим обсягом оперативної пам’яті. Обидва механізми керування пам’яттю – Anonymous SHared MEMory (ASHMEM) та Physical MEMory (PMEM) – додали новий спосіб виділення пам’яті для ядра. Ashmem може бути використаний для розподілу загальної пам’яті віртуальної пам’яті, а pmem дозволяє розподіляти фізичну пам’ять.

Anonymous SHared MEMory забезпечує іменовані блоки пам’яті, які можуть бути поділені між кількома процесами. На відміну від звичайної розподіленої пам’яті, ashmem може бути звільнений ядром. Для використання ashmem, процес відкриває файл /DEV/ashmem та виконує функцію mmap().

Physical MEMory (PMEM), наприклад, дозволяє драйверам або бібліотекам виділяти блоки фізично безперервної пам’яті. Цей драйвер був написаний для компенсації апаратних обмежень конкретного SoC – the MSM7201A.

Оптимізатор пам’яті Low memory killer є стандартним оптимізатором ядра Linux out of memory killer (oom killer) та використовує евристичні аналізатори для визначення «шкідливості» процесу, щоб мати можливість припинити його роботу. Вибираються процеси з найбільшою кількістю комірок з малою кількістю пам’яті. Така поведінка може бути незручною для користувача, оскільки oom killer може закрити поточну прикладну програму користувача, коли наразі існують інші процеси в системі, які не впливають на пам’ять.

Драйвер lowmemory починає свою роботу заздалегідь, до виникнення критичної ситуації нестачі пам’яті. Він виконує аналіз процесів та інформує їх про можливість закриття, для того, щоб процеси могли зберегти свій стан. Якщо ситуація з пам’яттю погіршується, lowmemory починає завершувати процеси.

4.1.4 API мобільних прикладних програм

У той час як більшість Android-прикладних програм написані мовою Java, є багато відмінностей між API Java та Android API. Основною відмінністю є те, що Android не використовує віртуальну машину Java, замість неї використовуються дві інші: Dalvik або Android Runtime (ART).

Android-платформа не містить віртуальної машини Java (Java VM), на якій виконується Java-байт-код. Замість цього класи Java компілюються у власний формат байт-коду, розроблений спеціально для віртуальної машини Dalvik. На відміну від віртуальних машин Java, які використовують стеки, Dalvik VM є архітектурою на базі регістрів.

Dalvik має деякі специфічні особливості, які відрізняють її від інших стандартних віртуальних машин: VM було розроблено, щоб використовувати менше пам’яті.

Пул регістрів був змінений, щоб використовувати тільки 32-бітові індекси для спрощення інтерпретатора.

Стандартний Java байт-код виконує 8-розрядні команди стека. Локальні змінні величини повинні бути скопійовані зі стека операндів з окремими інструкціями. Dalvik замість цього використовує свій власний 16-бітний набір команд, який працює безпосередньо на локальних змінних величинах. Локальна змінна величина зазвичай використовує поле в 4 біти «віртуального регістра».

Оскільки байт-код, який завантажується віртуальною машиною Dalvik відрізняється від байт-коду Java Virtual Machine, та в зв’язку з певним способом завантаження класів віртуальною машиною Dalvik, відсутня можливість завантажувати jar-файли. Інший порядок також повинен бути використаний для завантаження Android-бібліотеки. При цьому зміст базового DEX-файлу має бути скопійовано на карту пам’яті перед завантаженням.

Як і у випадку Java SE, Android клас System дозволяє витягувати властивості системи. Тим не менш, деякі обов’язкові властивості, визначені за допомогою віртуальної машини Java, не мають ніякого значення або мають інше значення в Android.

4.2 Архітектура платформи iOS

iOS є полегшеною версією настільної операційної системи Mac OS X. В основі Mac OS X лежить POSIX-сумісна ОС Darwin, Darwin – це ядро XNU, що з'явилося світ у результаті злиття мікроядра Mach і компонентів ядра FreeBSD.

Операційна система macOS, встановлена сьогодні на всі пристрої Apple, веде свою історію аж з 1988 року, який у світі IT відомий також тим, що став роком випуску першої бета-версії операційної системи NeXTSTEP. Сама NeXTSTEP була дітищем команди розробників Стіва Джобса, який на той час

уже залишив Apple і заснував компанію NeXT, яка зайнялася розробкою комп'ютерів для освітніх потреб.

У момент своєї появи на світ NeXTSTEP була передовою операційною системою, яка включала безліч технологічних новацій. В основі ОС було модифіковане мікроядро Mach, доповнене компонентами ядра FreeBSD, включаючи еталонну реалізацію мережевого стека. Більш високорівневі компоненти NeXTSTEP були написані з використанням мови Objective-C і надавали розробникам програм багатий об'єктно-орієнтований API. Система була забезпечена розвиненим і дуже зручним графічним інтерфейсом (ключові компоненти якого збереглися в OS X і навіть iOS) і потужним середовищем розробки, що включала в тому числі відомий всім сучасним розробникам візуальний дизайнер інтерфейсу.

Після провалу повернення Стіва Джобса до компанії Apple у 1997 році NeXTSTEP лягла в основу проекту Rhapsody, в рамках якого почалася розробка системи-спадкоємця Mac OS 9. У 2000 році з Rhapsody було виділено відкритий проект Darwin, вихідники якого опубліковані під ліцензією а вже в 2001 році з'явилася світ OS X 10.0, побудована на його основі. Через кілька років Darwin ліг в основу операційної системи для смартфона, що готується до випуску, про який до 2007-го, крім чуток, не було відомо майже нічого.

Умовно начинку OS X/iOS можна розділити на три логічні рівні: ядро XNU, шар сумісності зі стандартом POSIX (плюс різні системні демони/сервіси) та шар NeXTSTEP, що реалізує графічний стек, фреймворк та API додатків. Darwin включає в себе перші два шари і поширюється вільно, але тільки у версії для OS X. iOS-варіант, портований на архітектуру ARM і включає деякі доробки, повністю закритий і поширюється тільки в складі прошивок для айдевайсів (судячи з усього, це захист від портування iOS на інші пристрої).

За своєю суттю Darwin - це «гола» UNIX-подібна ОС, яка включає POSIX API, шелл, набір команд і сервісів, мінімально необхідних для роботи системи в консольному режимі і запуску UNIX-софту. У цьому плані він схожий на базову систему FreeBSD або мінімальну установку якогось Arch Linux, які дозволяють

запустити консольний UNIX-софт, але не мають ні графічної оболонки, ні всього необхідного для запуску серйозних графічних програм із середовищ GNOME або KDE.

Ключовий компонент Darwin - гібридне ядро XNU, засноване, як уже було сказано вище, на ядрі Mach та компонентах ядра FreeBSD, таких як планувальник процесів, мережевий стек та віртуальна файлова система (шар VFS). На відміну від Mach і FreeBSD, ядро OS X використовує власний API драйверів, названий I/O Kit і дозволяє писати драйвери на C++, використовуючи об'єктно-орієнтований підхід, що дуже спрощує розробку.

iOS використовує дещо змінену версію XNU, проте через те, що ядро iOS закрите, сказати, що саме змінила Apple, важко. Відомо тільки, що воно зібране з іншими опціями компілятора та модифікованим менеджером пам'яті, який враховує невеликі обсяги оперативної пам'яті в мобільних пристроях. У всьому іншому це все те ж XNU, яке можна знайти у вигляді зашифрованого кеша (ядро + всі драйвери/модулі) у каталозі /System/Library/Caches/com.apple.kernelcaches/ kernelcache на самому пристрої.

Рівнем вище ядра в Darwin розташовується шар UNIX/BSD, що включає набір стандартних бібліотек мови сі (libc, libmatch, libpthread і так далі), а також інструменти командного рядка, набір шеллів (bash, tcsh і ksh) і демонів, таких як launchd та стандартний SSH-сервер. Останній, до речі, можна активувати шляхом редагування файлу /System/Library/LaunchDaemons/ssh.plist. Якщо, звичайно, джейлбрейкнути девайс.

На цьому відкрита частина ОС під назвою Darwin закінчується, і починається шар фреймворків, які таки утворюють те, що ми звикли вважати OS X/iOS.

Фреймворки

Darwin реалізує лише базову частину Mac OS/iOS, яка відповідає лише за низькорівневі функції (драйвери, запуск/зупинка системи, керування мережею, ізоляція додатків тощо). Та частина системи, яка видна користувачеві та додаткам, до його складу не входить і реалізована у так званих фреймворках -

наборах бібліотек та сервісів, які відповідають у тому числі за формування графічного оточення та високорівневий API для сторонніх та стокових додатків

Як і в багатьох інших ОС, API Mac OS та iOS поділено на публічний та приватний. Стороннім програмам доступний виключно публічний і сильно урізаний API, проте jailbreak-додатки можуть використовувати і приватний.

У стандартній поставці Mac OS і iOS можна знайти десятки різних фреймворків, які відповідають за доступ до різних функцій ОС - від реалізації адресної книги (фреймворк AddressBook) до бібліотеки OpenGL (GLKit). Набір базових фреймворків для розробки графічних програм об'єднаний у так званий Cocoa API, свого роду метафреймворк, що дозволяє отримати доступ до основних можливостей ОС. У iOS він має ім'я Cocoa Touch і відрізняється від настільної версії орієнтацією на сенсорні дисплеї.

Не всі фреймворки доступні в обох ОС. Багато хто з них специфічний тільки для iOS. Як приклади можна навести AssetsLibrary, який відповідає за роботу з фотографіями та відео, CoreBlueTooth, що дозволяє отримати доступ до синезуба, або iAd, призначений для виведення рекламних оголошень у додатках. Інші фреймворки існують тільки в настільній версії системи, проте час від часу Apple переносить ті чи інші частини iOS в Mac OS або назад, як, наприклад, трапилося з фреймворком CoreMedia, який спочатку був доступний лише iOS.

Усі стандартні системні фреймворки можна знайти у системному каталозі /System/Library/Frameworks/. Кожен з них знаходиться у своєму власному каталозі, званому бандлом (boundle), який включає в себе ресурси (зображення та опис елементів інтерфейсу), хідери мови сі, що описують API, а також бібліотеку, що динамічно завантажується (у форматі dylib) з реалізацією фреймворку.

Одна з цікавих особливостей фреймворків – їхня версійність. Один фреймворк може мати відразу кілька різних версій, тому програма, розроблена для застарілих версій системи, продовжуватиме працювати, навіть незважаючи на зміни, внесені до нових версій ОС. Саме так реалізовано механізм запуску

старих iOS-додатків у iOS 7 та вище. Програма, розроблена для iOS 6, буде виглядати і працювати саме так, якби вона була запущена в iOS 6.

SpringBoard

Рівнем вище є програми, системні та встановлювані з магазину додатків. Центральне місце серед них займає, звичайно, SpringBoard (тільки в iOS), що реалізує домашній екран (робочий стіл). Саме воно запускається першим після старту системних демонів, завантаження в пам'ять фреймворків і старту дисплейного сервера (він же менеджер композитингу, він Quartz Compositor), що відповідає за виведення зображення на екран.

SpringBoard - це сполучна ланка між операційною системою та її користувачем, графічний інтерфейс, що дозволяє запускати програми, перемикатися між ними, переглядати повідомлення та керувати деякими налаштуваннями системи (починаючи з iOS 7). Але це також і обробник подій, таких як торкання екрана або переворот пристрою. На відміну від Mac OS X, яка використовує різні додатки та демони-агенти для реалізації компонентів інтерфейсу (Finder, Dashboard, LaunchPad та інші), в iOS майже всі базові можливості інтерфейсу користувача, у тому числі екран блокування та «шторка», укладені в одному SpringBoard.

На відміну від інших стокових додатків iOS, які розташовуються в каталозі /Applications, SpringBoard нарівні з дисплейним сервером вважається частиною фреймворків і знаходиться в каталозі /System/Library/CoreServices/. Для виконання багатьох завдань він використовує плагіни, що лежать у /System/Library/SpringBoardPlugins/. Крім усього іншого, там можна знайти, наприклад, NowPlayingArtLockScreen.lockboundle, що відповідає за відображення інформації про композиції, що програється на екрані блокування, або IncomingCall.serviceboundle, відповідальний за обробку вхідного дзвінка.

Починаючи з iOS 6 SpringBoard поділено на дві частини: сам робочий стіл та сервіс BackBoard, відповідальний за комунікації з низькорівневою частиною ОС, що працює з обладнанням (рівень HAL). BackBoard відповідає за обробку таких подій, як торкання екрана, натискання клавіш, отримання показання

акселерометра, датчика положення та датчика освітленості, а також керує запуском, призупиненням та завершенням програм.

SpringBoard і BackBoard мають настільки велике значення для iOS, що, якщо якимось чином їх зупинити, вся система застигне на місці і навіть запущена в даний момент програма не реагуватиме на торкання екрану. Це відрізняє їх від домашнього екрану Android, який є лише стандартним додатком, який можна зупинити, замінити або взагалі видалити з системи (у цьому випадку на екрані залишаться цілком робочі кнопки навігації і рядок стану зі «шторкою»).

Програми

На самій вершині цієї піраміди знаходяться програми. iOS розрізняє вбудовані (стокові) високопривілейовані програми та сторонні, що встановлюються з iTunes. І ті та інші зберігаються в системі у вигляді бандлів, багато в чому схожих на ті, що використовуються для фреймворків. Різниця полягає лише в тому, що бандл додатку включає дещо іншу метаінформацію, а місце динамічної бібліотеки займає виконуваний файл у форматі Mach-O.

Стандартний каталог зберігання стокових додатків – /Applications/. В iOS він абсолютно статичний та змінюється лише під час оновлень системи; користувач отримати доступ до нього не може. Сторонні програми, що встановлюються з iTunes, навпаки, зберігаються в домашньому каталозі користувача /var/mobile/Applications/ всередині підкаталогів, що мають вигляд 4-2-2-2-4, де два і чотири - це шістнадцяткові числа. Це так званий GUID – унікальний ідентифікатор, який однозначно ідентифікує додаток у системі та потрібен у тому числі для створення ізольованої пісочниці (sandbox).

Sandbox

В iOS пісочниці використовуються для ізолювання сервісів та додатків від системи та один від одного. Кожна стороння програма і більшість системних працюють у пісочниці. З технічної точки зору пісочниця є класичним для світу UNIX chroot, посиленим системою примусового контролю доступу TrustedBSD MAC (модуль ядра sandbox.kext), яка відрізає додаткам не тільки доступ до

файлів за межами домашнього каталогу, але й прямий доступ до заліза та багатьох системних функцій ОС.

Загалом укладений у sandbox додаток обмежений у таких можливостях:

- Доступ до файлової системи за винятком власного каталогу та домашнього каталогу користувача.
- Доступ до каталогів Media та Library всередині домашнього каталогу за винятком Media/DCIM/, Media/Photos/, Library/AddressBook/, Library/Keyboard/ та Library/Preferences/.
- Доступ до інформації про інші процеси (додаток «вважає» себе єдиним у системі).
- Прямий доступ до заліза (дозволено використовувати тільки Cocoa API та інші фреймворки).
- Обмеження використання оперативної пам'яті (контролюється механізмом Jatsam).

Всі ці обмеження відповідають sandbox-профілю (набору обмежуючих правил) container і застосовуються до будь-якої сторонньої програми. Для стокових додатків, у свою чергу, можуть застосовуватися інші обмеження, м'якіші або жорсткіші. Як приклад можна навести поштовий клієнт (профіль MobileMail), який має такі ж серйозні обмеження, як і сторонні програми, але може отримати доступ до всього вмісту каталогу Library/. Зворотна ситуація - SpringBoard, що взагалі не має обмежень.

Усередині пісочниць працюють багато системні демони, включаючи, наприклад, AFC, призначений для роботи з файловою системою пристрою з ПК, але що обмежує область видимості тільки домашнім каталогом користувача. Усі доступні системні sandbox-профілі розташовуються в каталозі /System/Library/Sandbox/Profiles/* і є наборами правил, написаних мовою Scheme. Крім цього, додатки також можуть включати додаткові набори правил, званих entitlement. По суті, це все ті ж самі профілі, але вшиті прямо в бінарний файл додатка (своєрідне самообмеження).

Сенс існування всіх цих обмежень подвійний. Перше (і головне) завдання, яке вирішує sandbox, - це захист від шкідливих програм. Разом з ретельною перевіркою опублікованих у iTunes додатків та забороною на запуск не підписаних цифровим ключем додатків (читай: будь-яких, отриманих не з iTunes) такий підхід дає чудовий результат і дозволяє iOS перебувати на вершині у списку найзахищеніших від вірусів ОС.

Друга проблема - це захист системи від самої себе та користувача. Баги можуть існувати як у стоковем софті від Apple, так і в головах користувачів. Sandbox захищає від обох. Навіть якщо зловмисник знайде дірку в Safari і спробує її експлуатувати, він все одно залишиться у пісочниці та не зможе нашкодити системі. А користувач не зможе «зламати свій улюблений телефон» і не напише гнівних відгуків на адресу Apple. На щастя, знаючі люди завжди можуть зробити jailbreak і обійти захист sandbox (власне, в цьому є сенс джейлбрейка).

Багатозадачність

Одна з найспірніших особливостей iOS – це реалізація багатозадачності. Вона начебто і є, а з іншого боку, її немає. У порівнянні з традиційними настільними ОС і горезвісним Android iOS не є багатозадачною операційною системою у звичному значенні цього слова і не дозволяє програмам вільно працювати у фоні. Натомість ОС реалізує API, який програма може використовувати для виконання окремих завдань, поки вона знаходиться у фоновому режимі.

Вперше такий API з'явився в iOS 4 (до цього фонові завдання могли виконувати тільки стокові програми) та нарощувався у міру розвитку операційної системи. Сьогодні (йдеться про iOS 7) так званий Background API дозволяє робити наступне:

- програвати аудіо;
- здійснювати VoIP-дзвінки;
- отримувати інформацію про зміну розташування;
- отримувати push-сповіщення;

- планувати відкладений висновок повідомлень;
- вимагати додатковий час для завершення роботи після переходу у фоновий режим;
- обмінюватись даними з підключеними до девайсу аксесуарами (у тому числі Bluetooth);
- отримувати та надсилати дані по мережі (починаючи з iOS 7).

Такі обмеження на роботу у фоні необхідні в першу чергу для того, щоб зберегти заряд батареї та уникнути лагів інтерфейсу, так знайомих користувачам Android, де програми можуть робити у фоні все, що захочуть. Насправді Apple настільки сильно дбає про збереження батареї, що навіть реалізувала спеціальний механізм для угрупування фонових дій додатків та їх запуску в потрібні моменти, наприклад, коли смартфон активно використовується, підключений до Wi-Fi-мережі або до зарядного пристрою.