ОПЕРАЦИОННА СИСТЕМА ANDROID - СТРУКТУРА, ОРГАНИЗАЦИЯ, АРХИТЕКТУРА

Д-р инж.Мартин Пъшев Иванов,
главен асистент, НБУ –департамент „Информатика”,
e-mail: mivanov@nbu.bg

Анотация: В настоящето изложение са описани фундаменталните принципи на изграждане и функциониране на операционната система Android. Описан е детайлно многослойният архитектурен модел на операционната система, представени са съществените ѝ страни и характеристики като управление на процесите,организация и управление на паметта, файлова система, принципи на сигурността и др.

Ключови думи: *OS Android, архитектура, управление на процесите*

1. **Принципи на изграждане**.

Операционната система Android към настоящия момент се е наложила като най-широко разпространената и най-предпочитаната операционна система за мобилни устройства с ограничена изчислителна мощност и ресурси. Успешното ѝ развитие и доказаните ѝ предимства водят до разширяване на нейния обхват на приложимост и за кратко време тя получава популярност и върху други класове устройства – интелигентни телевизори, часовници и т.н. Тя е с отворен код, а исторически началото на разработването ѝ попада някъде около 2003 г., когато четирима млади ентусиасти основават компанията Android Inc., базирана в Пало Алто, Калифорния. Младият екип си поставя задача да разработи операционна система за мобилни устройства, която да отчита личните предпочитания и особености на потребителя, както например географското му местоположение. Целевият клас усройства първоначално е цифрови фото- и видеокамери, които да могат да могат да ползват определени компютърни услуги, но впоследствие разработчиците разширяват пазарния сегмент на изделието си, като фокусират вниманието си в областта на мобилните комуникации. През август 2005 г. фирмата Android Inc. е закупена от Google, с което компанията прави сериозна заявка да развива дейност в областта на Web-базирани услуги за мобилни платформи. Решаващ етап от развитието на концепцията за мобилната операционна система е създаването на консорциума *Open Handset Alliance* през ноември 2007 г. от няколко компании, създатели на мобилни компютърни технологии и доставчици на мобилни комуникации, сред които *Google, HTC, Motorola, Intel, Qualcomm, Sprint Nextel, T-Mobile*, и *NVIDIA* . Тези компании си поставят като цел разработването на отворени стандарти за мобилни устройства и обявяват своя нов продукт – Android – платформа за мобилни платформи, базирана на *Linux Kernel*, която ще бъде отворена и достъпна за широк клас мобилни устройства и за производителите им. Като софтуер с отворен код *Android* се появява през октомври 2008 г.

Позиционирането на ОС Android на пазара на мобилните устройства е повече от оптимистично. Някои кратки данни: през третото тримесечие на 2014 г. общия брой продадени мобилни устройства, базирани на Android надхвърля 268 милиона, значително превъзхождащ съответния показател на основния си конкурент iPhone. Към края на 2014 г. продажбите на Android-устройствата вече надхвърлят милиард. През второто тримесечие на 2014 г. Android контролира пазарен дял от 84.7 % от общото производство на смартфони, изпреварвайки далеч iPhone, Windows Phone и BlackBerry. Android категорично доминира в областта на таблетите – около 62 % от близо 195 милиона таблети, продадени през 2013 г. са устройства с ОС Android.

Операционната система, чийто обхват включва широк диапазаон устройства с различна архитектура и с различна изчислителна мощност, е обект на периодични обновявания, които имат за цел отстраняване както на технически проблеми, така и разширяване на функционалностите и услугите, предоставени на потребителя. След началната реализация на смартфона HTC Dream (известен и като T-Mobile G1), ОС Android получава общо 19 основни актуализации. Периодът на актуализация на Android е средно около два месеца и половина, значително по-бързо от цикъла на разработване на iOS (чиито основни актуализации са веднъж годишно). Всяка версия получава свое собствено кодово име, свързано с някакъв вид десерт. Кодовите имена се присвояват в азбучен ред.

Последната важна актуалзация на OS Android е версия 5.1 – 5.1.1 (API Level 22), известна под името Lollipop. Същата е въведена през м.март-април 2014 г. Актуализацията обхваща поддръжка на 64-битови процесори, удължен живот на батерията, регистрация на потребител с права на „гост", модернизиран графичен потребителски интерфейс и др. Променена е съществено и концепцията на изпълнителната среда за Java-приложения. Традиционната за предишните версии на системата Java-виртуална машина Dalvik е заменена със средата ART (Android RunTime), а механизмът JIT (Just In Time) - с компилация на байт-кода при инсталиране на приложението (наречена Ahead-Of-Time – AOT компилация).

Android е истинска отворена безплатна платформа за разработване предимно на мобилни приложения. Тя се радва на нарастваща популярност сред производителите на мобилни устройства, защото те могат да я ползват и приспособяват, без да имат никакви финансови ангажименти към създателите ѝ. Нейно значително предимство е, че съществуването ѝ не зависи от определен доставчик, чието присъствие на пазара на специализиран софтуер може да е времено.

Android е пълноценна операционна система, която предоставя поддръжка на управлението на процесите и паметта, файлова система, драйвери за разнообразни устройства и мрежови комуникации. В основата си Android се основава на ядрото на ОС Linux – също с отворен код. В ранните реализации на Android това е Linux kernel 2.6 (2.6.25 за Android 1.0), а от април 2014 г. това са основно версиите 3.4 и 3.10 на това ядро. Развитието на версиите на ядрото зависи от развитието и състоянието на поддържаните от тях хардуерни платформи. Базовият компонент kernel е приспособен за нуждите на конкретния начин на употреба и условията на работа на системата – той бива допълнен и непрекъснато разширяван с някои нови подобрения, други функционалности отпадат или се видоизменят съществено. Така например Android не поддържа традиционните за Linux техники за комуникация между процесите и ги заменя с механизма, предоставен от драйвера Binder. Допълнително в ядрото на операционнната система се включват компонентите "low memory killer", "logger", „ashmem" (Anonymous SHared MEMory), компонент за управление на консумираната енергия (power management) и др.

Android се отличава с компонентно базирана архитектура, повлияна от разнообразието в *Internet*: части от едно приложение могат да бъдат използвани от друго, при това по начин, различен от първоначално планирания. Могат да бъдат заместени дори и вградените компоненти на средата със свои собствени усъвършенствани версии. Това разширява обхвата на творческия процес в областта на мобилния софтуер

Android е многозадачна система. Тя дава възможност за едновременното изпълнение на няколко системни и потребителски процеси, както и за поддържането на множество нишки (олекотени процеси) във всеки от тях. Всеки от процесите функционира в собствено виртуално адресно пространство, като работата му е напълно изолирана от работата на другите процеси – т.е.той няма достъп до техните данни и ресурси, както и те до неговите.

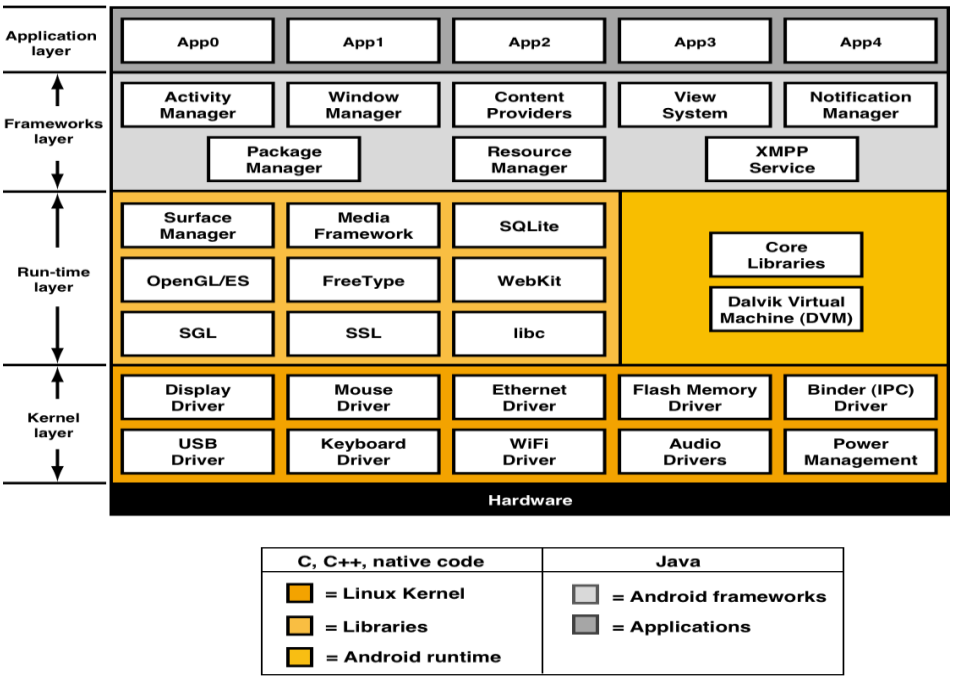
Android е ориентирана предимно към мобилни устройства за постоянна и ежедневна употреба. Като такава тя позволява поддържането на потребителски интерфейс с директна манипулация – различни видове сензорни екрани на мобилни телефони и таблети.

Преобладаващите хардуерни платформи, към които е ориентирана Android, имат ARM-архитектура (ARMv7 или по-късна, Android 5.0 също така поддържа и ARMv8-A). Официално се поддържат и архитектури от типа x86 и MIPS. За всички отбелязани платформи от реализацията на Android 5.0 се поддържат както 32-битовите, така и 64-битовите версии. Следва да се отбележи, че от 2012 г. в обхвата на Android започват да се включват и устройства с процесори на Intel, сред които има телефони, таблети и др.

От ноември 2013 Android 4.4 препоръчва наличието на поне 512 MB RAM памет в мобилното устройство. За устройствата с ограничена памет („low RAM") се изисква наличие на минимум 340 MB, която не включва паметта, предназначена за различни хардуерни компоненти, като например "baseband" процесори. Android 4.4 изисква процесори с 32-битови ARMv7, MIPS или x86 архитектура, заедно с графичен процесор (graphics processing unit - GPU), съвместим с OpenGL EC 2.0. По принцип Android поддържа OpenGL ES 1.1, 2.0 и 3.1. Някои приложения могат да изискват допълнително и наличието на OpenGL ES, както и на подходящ графичен хардуер.

2. **Архитектура на Android**.

Android има модулна организация и стекова архитектура ([1], [2], [3]). Компонентите на стековата архитектура са разпределени в четири основни слоя, всеки от които поема специфични функции, свързани с работата на операционната система и изпълняваните потребителски приложения. Това са слоевете: приложен слой, application framework, среда на изпълнение за Java-приложения и библиотеки , слой на ядрото на Android (Linux kernel). Всеки слой осигурява функционирането на слоевете над него в стека и същевременно ги изолира от детайлите, характеризиращи работата на слоевете от по-ниско ниво. Стековата организация на Android е изобразена на фиг. 1 .



Фиг. 1 . Стекова организация на архитектурата на ОС Android (Източник: Freescale).

По-долу са изброени основните характеристики на слоевете в стековата организация на ОС Android.

1.1 Приложен слой

Приложният слой е разположен на върха на софтуерния стек и е разработен изключително на Java. Тук са включени по подразбиране системни приложения, използвани от потребителя като: Browser, Phone, Contacts, Gallery, Calendar и др., които се разпространяват с Android. Този слой включва и инсталираните в средата потребителски приложения, набавени например от Google PlayStore или дори разработени самостоятелно. Приложенията от този слой използват компонентите и услугите, предоставени от слоя Application Framework и се изпълняват чрез виртуалната машина Dalvik (DVM), а от Android 4.4 (KitKat) – в изпълнителната среда ART.

1.2 Слой „Application Framework" (Софтуерна рамка на приложенията)

Този слой включва съвупност от услуги, които съвместно формират средата, в която приложенията за Android се изпълняват и се управляват. Архитектурата на тези компоненти е разработена с оглед опростяването на многократното им използване, а за написването им е използван изцяло езикът Java. Тук се разполагат основните услуги на Android за управление на жизнения цикъл на приложенията, на пакети, ресурси т.н. Всяко едно приложение може да използва базовите възможности на съставляващите Application Framework както директно, така и индиректно - чрез средствата и разрешенията, предоставени от други приложения. Програмистите имат пълен достъп до тези компоненти чрез съответния API (представен в придружаващите средата Java пакети и класове).

Основните услуги, ключови за разработването на приложения под Android, съдържащи се в този слой са:

- **Activity Manager** – отговорна за всички аспекти, свързани с протичането на жизнения цикъл на приложението (стартиране, спиране, възстановяване на изпълнението и т.н.) и поддържания софтуерен стек от activity (activity backstack).
- **Window Manager** – представляващ Java абстракция на поддържания в системата мениджър на екрана (surface manager). Window Manager осигурява достъп до функците на мениджъра на екрана, като дава възможност на приложението да обяви своето клиентско визуално пространство и да използва някои екранни елементи като например лента на състоянието (status bar).
- **Content providers (Доставчици на съдържание)** – позволяващи на приложенията да публикуват и споделят данните си с други приложения.
- **Notification Manager** – отговорен за уведомяването на потребителя за настъпили събития (често - при изпълнението на фонови задачи).
- **Resource Manager** – предоставящ достъп до вътрешните ресурси на приложението, които не са програмен код – символни низове, цветове, параметри на оформлението на графичния интерфейс, битови карти и др.
- **View System** – разширяема система от визуални компоненти и характеристик, иползвани за изграждане на потребителския интрфейс на приложенията и за комуникация с потребителя.
- **Package Mananger** - система от услуги за инсталиране и деинсталиране на приложения, както и за получаване на информация за приложенията, текущо инсталирани на устройството.
- **Telephony Manager** – предоставя достъп и информация за телефонните услуги, достъпни на устройството, като например информация за статуса и абоната, SMS, MMS и др.

- **Location Manager** – предоставя достъп до услугите за местоположение и до информацията, свързана с актуализирането на промените в местоположението – чрез GPS, клетъчни ID или чрез локални WI-Fi бази от данни.

1. 3 Java-изпълнителна среда (Dalvik VM или ART) и обслужващите я библиотеки. Библиотеки, съдържащи и обслужващи средата за изпълнение

В този слой влизат: виртуалната машина Dalvik (респ.наследяващата я изпълнителна среда ART) и специфични за работата ѝ библиотеки, някои важни библиотеки, чиито модули са написани на C и C++ и са компилирани в конкретния машинен код (за процесори ARM или Intel86) и библиотеките от Java Native Interface.

Изпълнителната среда (runtime), е тази, която непосредствено осигурява изпълнението на едно Java-приложение. Всяко такова приложение се изпълнява в своя изпълнителна среда, като многозадачната работа на приложенията се осигорява чрез средствата, предоставени чрез функционалностите на ядрото на Linux. Изпълнителната среда включва виртуалната машина (или ART за поседните версии на Android), заедно с необходимите за функционираните ѝ библиотеки (Core Libraries или Dalvik Libraries). Последните се разделят на три категории:

- Специфични за Dalvik VM библиотеки, които се ползват от приложенията за да взаимодействат пряко с нейните инстанции и рядко се ползват от разработчиците и от програмистите.
- Java-библиотеки на оперативната съвместимост (Java Interoperability Libraries) – приложенията за Android са разработени предимно на Java. Стандартната среда за разработване на Java-приложения включва широкообхватно множество от класове и интерфейси, представени в основните работни библиотеки (пакети). Тези библиотеки предоставят средства за обработка на символни низове, работа в мрежа, файлов вход/изход и др. Оперативните Java-библиотеки от този слой са функционален аналог (с отворен код) на подмножество на стандартните Java-библиотеки (включени в редакцията Java SE), които са адаптирани и трансформирани за употреба от приложенията, работещи в средата на Dalvik VM.
- Библиотеки на Android OS – тази категория обхваща тези Java-базирани библиотеки, които са специфични за разработването на приложения в Android. Повечето от тези библиотеки са представни чрез съдържанието на пакетите **android.app, android.content, android.database, android.graphics, android.hardware, android.text, android.widget, android.webkit, android.provider и др.**

Една от особеностите на този слой е поддържането на функционалност, която позволява на приложенията, написани на Java и работещи в изпълнителната среда (Dalvik или ART) да ползват модули и системни библиотеки, съдъжащи процесорно-зависими операции и получени от комилация на C и C++ код (както и обратното – от програми на C или C++ да се извикват методи от Java-класове). Тази функционалност се осигурява от компонента Java Native Interface (JNI). Функциите на този компонент правят възможно от Java байт-код да бъдат изпълнявани обръщения към специфични системни функции и хардуерни драйвери от по-ниските равнища на софтуерния стек на ОС. Следва да се отбележи, че това полезно и необходимо разширение влече след себе си някои проблеми, свързани както със загуба на преносимостта и с намаляване на сигурността на Java-кода, така и със загуба на обикновено изпълняваните в Java кода вътрешни проверки на типове и стойности на променливи и граници на масиви.

Най-съществените причини за използване на JNI в кода са следните:

- **Ефективност** - в някои специфични случаи на работа на графични програми или на приложения, интензивно натоварващи процесора (напр.видео-декодиране) JNI позволява да се използват специфични за процесора възможности, недостъпни за изпълнителната среда (Dalvik или ART);
- **Обфускация** – някои разработчици предпочитат тази техника, за да предпазят кода си от дизасемблиране (което в случаите на изплзване на Java-байт код в .dex файловете е лесно осъществимо). В такива случаи основното приложение се разпространява заедно с библиотека съдържаща значителна част от бизнес-логиката и затрудняваща възстановяването му в декомпилиран вид.

1.4 Системни и функционални библиотеки на C и C++.

Това равнище включва множество библиотеки на C/C++, компилирани в специфичния за хардуера код (native код) и използвани от различни компоненти на операционната система. Библиотеките от това равнище могат да бъдат разделени според своето функционално предназначение и мястото им в софтуерния стек на Android на две основни групи:

- Системна библиотека на C;
- Функционални библиотеки на C/C++.

Системната библиотека се основава на версията Berkeley Software Distribution (BSD). Компанията Goolge е разработила собствена версия на системната библителa libc – Bionic специално за мобилни устройства на основа Linux. Това е породено от необхдимостта да се постигне ограничение на използваната памет и по-бързо зареждане на библиотеките във всеки процес, поради което е преследвана целта тази библиотека да има минимален размер. Библиотеката Bionic обхваща обем около 200 kB, което е около два пъти по-малко от размера на стандатната за Linux библиотека glibc. При това се взема под внимание и ограниченият изчислителен потенциал на мобилното устройство, т.е.библиотеката е оптимизирана и по отношение постигането на максимално бързодействие., особено за по ранните версии на Android, разпространени върху устройства със значително по-скромни хардуерни характеристики.

Библиотеката Bionic има вградената поддръжка на важни за системата Android системни услуги и регистрация на системни събития, но паралелно с това не поддържа някои функционалности – например изключения на C++ и е несъвместима с GNU libc и със стандарта POSIX.

Функционалните библиотеки са разработени на C/C++ . Тук влизат библиотеките:

- WebKit – съдържа софтуерен механизъм за представяне и изобразяване на web-страници в барузъра. WebKit съдържа възможности за подобряване на работата на браузърите Apple Safari и Google Chrome.
- Media Framework – медийна библиотека, поддържаща функции за работа с повечето аудио- и видеоформати., включващи MPEG-4, JPG, PNG, H.264. и AAC.
- SQLite – пълноценна „олекотена” версия на система за управление и поддържане на релационни бази от данни, които се изпозват от приложенията.
- OpenGL – съвкупност от графични библиотеки.
- SGL (Scalable Graphics Libraries) – за изобразяване на двумерни графики.
- OpenSSL – инструментариум с отворен код , който реализира Secure Socket Layer (SSL).

Повечето от тези библиотеки са с отворен код и могат да бъдат използвани без каквито и да е изменения. За разработчиците достъпът до функциите на тези библиотеки се реализира чрез изплзването на слоя Application Framework. Библиотеките са достъпни от Java-код чрез съответните интерфейсни механизми (JNI).

1.5 Слой на ядрото на ОС Android (Linux Kernel)

Базовата подсистема в стековата организация на ОС Android – Linux-ядрото е тази, която осигурява фундаметалните функции на взаимодействие на системния софтуер с конкретната хардуерна конфигурация. Тази подсистема е отговорна така също и за най-типичните функции на операционната система като управление на процесите и паметта, многозадачност, файлова система, входни-изходни канали, системни услуги от ниско ниво, управление на захранването и т.н. Ядрото на ОС е мястото, където са реализирани и инкорпорирани в системата специфичните за хардуерната платформа драйвери, осигуряващи възможност за работа с разнообразието от съществуващи хардуерни компоненти - Wi-Fi и Bluetooth комуникации, сензорни екрани, камери, акселерометри, GPS-приемници и др. Слоят на ядрото е основа на цялата стекова архитектура на ОС Android, която е създадена така, че да предоставя максимална гъвкавост и възможност за работа с огромното разнообразие от хардуерни мобилни конфигурации и модели устройства.

Във функциите на тази подсистема (слой) влиза и осигуряването на т.нар HAL – Hardware Abstraction Layer (представляващ фактически „вододела" между хардуерната и софтуерна части), което се предоставя от наличните в библиотеката libhardware.so средства . Целта е да се „абстрахира" функционирането на горните слоеве на архитектурния стек от значителното разнообразие и специфични хардуерни решения, изплзвани в мобилните устройства. Наличието на HAL спомага за стандартизацията в слоя на ядрото, като изисква от доставчика на устройството да постави съответния адаптер в системната директория /system/lib/hw , при което libhardware.so ще го намери и ще го зареди автоматично.

Android притежава няколко много важни разширения на управлението на паметта (Memory Management Extensions), които не са налични в стандартното Linux-ядро. Първото е компонента ASHMem (Anonymous SHared MEMory) – механизъм за анонимно споделяне на паметта, който представя абстрактно споделената памет под формата на файлови дескриптори. Този механизъм се реализира от модула **`mm/ashmem.c`** и се използва много активно.

Друг съществен механизъм се предоставя от компонента Pmem, който е отговорен за разпределението на на физически непрекъснато пространство памет. Това е необходимо в някои случаи, когато хардуерът не поддържа виртуална памет или не може да извърши логическо обединение на няколко отделни области от паметта (напр.камерата на логическото устройство).

Разширението Low Memory Killer представлява механизъм, реализиран на базата на OOM (out-of-memory) механизма на Linux (Linux Kernel 2.6.67). Необходимостта от такъв механизъм възниква поради това, че обикновено мобилните устройства нямат възможност за „swap" на паметтта и когато физическата памет се окаже недостатъчна за някое от приложенията, то неговата работа просто се прекратява (приложението се „убива"). Прекратяването на работата на приложението става само в случаите, когато неговата работа е изолирана от работата на други приложения.

Компонентът binder има особено значение за поддържания от Android механизъм за междупроцесна комуникация (IPC – Interprocess Communication). Той осигурява използвания от средата за изпълнение механизъм „AIDL" (Android Interface Definition Language) за междупроцесна комуникация чрез специфичен драйвер, предоставен от ядрото.

Подсистемата за регистриране (logging subsystem) позволява поддържането на отделни регистрционни (log) файлове за различните подсистеми на Android. Регистрационните файлове са достъпни в потребителски режим в каталога **`dev/log`** .

Компонентът RAM Console е разширение, което позволява на ядрото при непредвидени аварийни ситуации да изведе данни в RAM-паметта на устройството. В стандартната Linux система това става в специален обменен (swap) файл, но мобилните устройства поради ограничените си реурси нямат тази възможност.

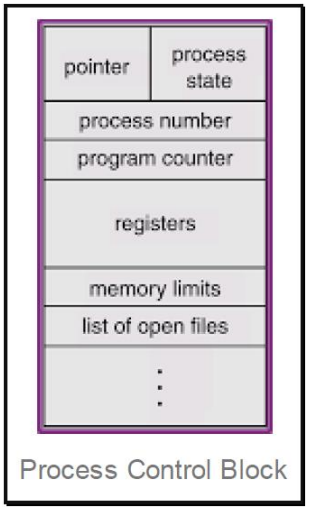
Компнентите Wakelocs и alarms се две вградени в Android разширения на компонента за управление на захранването.

2. Процеси и управлението им в Android

Стартирането на всяко приложение в Android е свързано със създаването на отделен процес, който първоначално съдържа една единствена изпълнявана нишка. По подразбиране всички компоненти на приложението се изпълняват в тази нишка, която се нарича „главна" (main). Възможно е организирането на изпълнението на отделните компоненти на приложението както в отделни нишки, така и в отделни процеси. Възможно е и асоцииране на отделен компонент на приложението към конкретна нишка , което се указва в манифестния файл на приложението (чрез атрибута **android:process** на елемента **<application>**). Възможно е дори изпълнението на компоненти на различни прилжения в един и същи процес, при условие, че приложенията споделят един и същи потребителски ID и са подписани с един и същ сертификат.

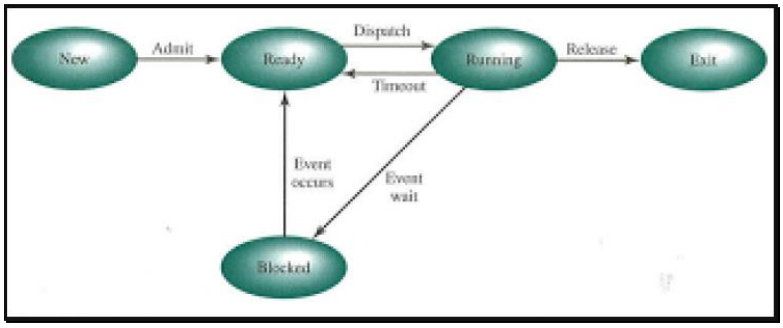
Процесите в ОС Android са по същество Linux-процеси. В Linux се реализира специфичен механизъм на виртуалната памет, който предоставя на всеки процес достъп до едно линейно и непрекъснато свързано виртуално адресно пространство. Това непрекъснато пространство от виртуални адреси се изобразява върху физическото адресно пространство на операционната система. Всеки процес работи в своето виртуално адресно пространство и няма достъп до адресните пространства на останалите процеси – т.е. достъпът на процесите до паметта се ограничава до техните собствени виртуални адреси. Само операционната система има достъп до адресното пространство, представяно от физическата памет.

Управлението на процесите в Android в общи линии се поддържа от характерните за повечето операционни системи техники и структури от данни. Типичният блок за управление на процесите за Android е показан на фиг. 2 .



Фиг.2 Типична структура на блок за управление на процесите в Android.

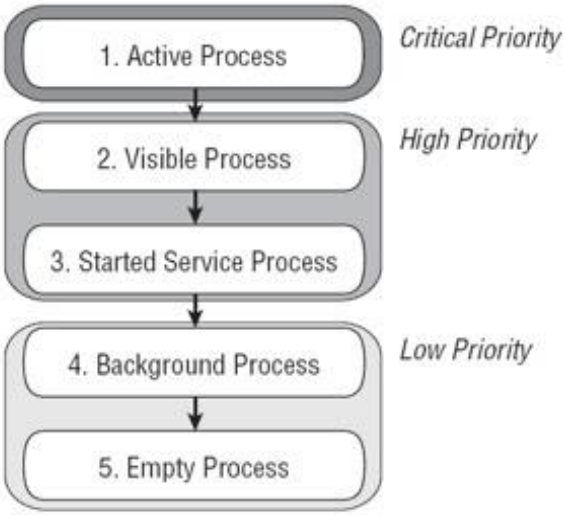
Схемата на управление на процеса в Android също не се различава съществено от стандартната за операционните системи (фиг.3):



Фиг. 3 . Състояния на процеса и преходи между състоянията в Android.

Процесите в Android притежават приоритети, базирани на системата от приоритети, приета в ОС Linux (със стойности от -20 до +19). Приоритетите на процесите се установяват при тяхното стартиране от компонента на ОС System server и се използват при диспечирането им от системата. Всеки процес включва поне една изпълнявана нишка, която се стартира със стартирането на процеса. Отделните нишки в процеса също имат свой собствен приоритет (същият може да бъде установен с метода **`Process.setThreadPriority(...)`**).

Друг тип оперативни приоритети организира процесите в Android в йерархия “по важност”. Тази приоритетна организация се използва в случаите, когато при необходимост и поради липса на ресурси в системата някой от процесите трябва да бъде преустановен. Йерархията и „важноста” на процеса в нея се определя от това какви компоненти се изпълняват в него и какво е тяхното текущо състояние. Компонентите, които са най-ниско в йерахията се елиминират първи. Съществуват пет равнища на йерархията на значимост, според състоянието на компонентите на приложението и подредени по посока на намаляване на „важността” на процесите (фиг. 4):



Фиг. 4 . Състояния и приоритети на процесите.

- **Активни (foreground) процеси** - Процес, който се изисква от текущите действия на потребителя. Такъв процес е в състояние да поддържа активно взаимодействие с потребителя. Към неговото прекратяване се пристъпва само в крайни, изключителни случаи. Процесът е *активен* ако е налице някое от следните условия:

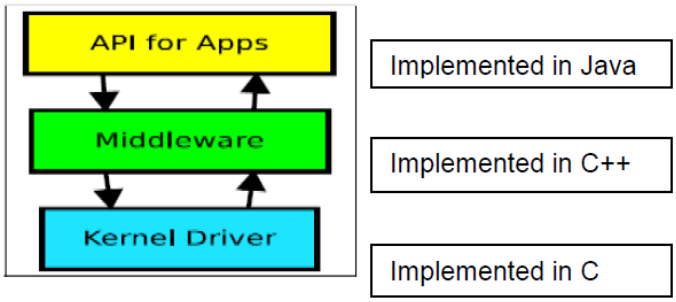
    - Съдържа *activity*, с което потребителят взаимодейства (извиква се методът **onResume()** на **Activity**).
    - Съдържа *service*, която е свързана с **Activity**, с което потребителя взаимдейства.
    - Съдържа *service*, която се изпълнява "във foreground" – изпълнен е метода **startForeground()** на **Service**.
    - Съдържа *service*, която изпълнява някой от методите на жизнения си цикъл (**onCreate()**, **onStart()**, **onDestroy()**).
    - Съдържа **BroadcastReciever**, който изпълнява своя **onRecieve()** метод.
- **„Видими" (*Visible*) процеси** - Процес, който няма *foreground* компоненти, но все още може да възприеме това, което потребителят вижда на екрана. Процесът се отбелязва като *visible*, ако е налице някое от следните условия:
    - Съдържа *activity*, което не е *foreground*, но остава видимо за потребителя (извикан е методът му **onPause()**).
    - Съдържа service, която е свързана към *visible* или *foreground activity*.
- **Процеси - „услуги" (*service processes*)** - Процес, в който се изпълнява услуга, която е стартирана с метода **startService()** и не попада в горните две категории. Въпреки, че сървисните процеси не са свързани с нищо, което да вижда поребителят, те като цяло правят това, което последният изисква (музикален плеър, сваляне на данни). Системата запазва такива процеси, освен ако не е необходимо да се освободи памет за процеси от по-горните приоритетни равнища.
- **Фонови (*Background*) процеси** - Процес, съдържащ *activity*, което не е текущо видимо за потребителя (извикан е методът **onStop()** на *activity*). Тези процеси нямат директно въздействие върху работата на потребителя и системата ги прекратява всеки път, когато е необходимо да се освободи памет за *foreground*, *visible* или за *service* процеси.
- **„Празни" (Empty) процеси** - Процес, който не съдържа никакви компоненти на приложението. Основната причина да се поддържа "жив" такъв процес е за нуждите на кеширането и за да се ускори времето за стартиране на следващия компонент. Системата обикновено прекратява тези процеси, за да балансира общите системни ресурси между кеша на процеса и кеша на ядрото.

2.1 Комуникация между процесите в Android

Както бе отбелязано неколкократно, приложенията в Android се изпълняват в изолирани процеси и за случаите, когато е необходимо те да комуникират помежду си чрез обмен на данни или пък да ползват системните услуги, е необходим специализиран механизъм, известен като "InterProcess Communication" (IPC). Въпреки, че Android поддържа всички традиционни техники за междупроцесна комункация, реализирани в ядрото на Linux (напр.канали, съобщения, семафори, сокети, споделена памет и т.н.), те не се използват в библиотеките на OS и в платформените програмни интерфейси (API). Вместо тях Android използва свой собствен механизъм, обхващащ слоя на Linux-ядрото, междинната фаза на софтуерния стек и приложния слой – т.нар софтуерна рамка Binder. Изплзването на Binder от Android се налага от преди всичко от съображения за подобрена производителност.

Binder се определя като инструментален компонент от ниско ниво, който улеснява надеждните синхронни комуникации между процесите, изпълнявани в различни, изолирани по между си среди. Действието му се основава на използването на споделена памет в областта на на Linux-ядрото, посредством която се извършва обменът на данните между комуникиращите поцеси. В рамките на структурата на ядрото на операционната система функционалността на Binder се поддържа от специален драйвер – binder, чиято роля е да изобрази адресите от паметта, които изплзва всеки процес върху адресното пространство на ядрото.

Софтуерната рамка (framework) Binder следва комуникационния модел „клиент/сървър” и е реализирана в три отделни равнища на абстракция. Всички Java приложения имат достъп до функционалностите на binder чрез специализиран интерфейс (Binder), представен като равнище в програмния интерфейс на Java (равнище API for Applications). Java-реализцията на интерфейса Binder на свой ред ползва интерфейса, реализиран на равнището на междинната фаза (състояща се от библиотеки, написани на C++). Достъпът до билиотеките на междинната фаза се осигурява от JNI. Междинната фаза от своя страна е отговорна за маршализацията и демаршализацията на обектите (референтните типове) и за комуникацията с binder-драйвера от ядрото, като се използва комуникационен протокол от ниско ниво (фиг. 5).



Фиг. 5 . Архитектура на софтуерната рамка Binder.

Организацията на функциите на Binder в три абстрактни равнища освобождава програмистите от ангажимента по разработването на детайлите от по-ниските нива на комуникацията между процесите, след като основното (най-високо) равнище поема изцяло грижата за обслужване на всички заявки за междупроцесна комуникация. Най-важното усъвършенстване в този механизъм от гледна точка на програмистите е, че методите на „отдалечените” обекти могат да се извикат като локални обръщения. Това се постига чрез синхронно извикване на методите, при което клиентският процес се блокира за времето до получаване на отговора от сървърния процес.

Механизмът, заложен в Binder позволява удобна и ефективна комуникация извън границите на отделните процеси, доколкото същите не могат да споделят съдържанието на виртуалните си машини. Този механизъм не е идеален и една от неговите слабости е, че не е подходящ за обмен на големи потоци от данни (напр.аудио или видео-потоци) доколкото всеки обект преди пренасянето следва да бъде преобразуван в съответен вид, а след това възстановен (класът Parcel).

Механизмът на междупроцесна комуникация ще бъде разгледан по-детайлно и в необходимата за изложението пълнота в глава ... .

3. Организация на адресното пространство

Както във всички съвременни операционни системи, достъпното адресно пространство при OS Android се разделя на две отделни области:

- Потребителско пространство (User Space), в което се разполагат и изпълняват потребителските програми и данни;
- Пространство на ярото (Kernel Space), в което се разполага и изпълнява кода на ядрото.

Потребителските процеси нормално се изпълняват в непривилегирован режим, което означва, че те нямат непосредствен достъп до физическата памет или до устройствата. Разположените в пространството на ядрото процеси имат пълен достъп до физическата памет и

до устройствата и се изпълняват в режим, определен като „привилегирован” или като „режим на ядрото” (kernel mode). Изпълняваните в потребителското пространство процеси имат достъп до някои от функциите на ядрото чрез предоставения им програмен интерфейс – т.нар. „системни извиквания” (system calls). Коцепцията за разделяне и защита на потребителското пространство и пространството на ядрото се основава на възможност, поддържана от хардуерната платформа и обозначена с понятието „пръстени” (rings), всеки от които обединява група права. Типична за архитектурата на Intel x86 е организацията на защитата в четири „пръстена”, но в Linux и подобните ѝ системи се използват само първия – с най-високи права (ring 0), определен като „kernel mode” (режим на привилегировани операции) и последния – с най-ниски права (ring 3), в който се изпълняват потребителските процеси.

4. **Особености на файловата система в Android**.

По принцип файловата система в OS Android е твърде близка до тази, характерна за OS Linux. Във файловите системи, присъщи на Linux и на подобните на нея ОС файловата йерархия е единично дърво с корен, отбелязван като „/” (“root”). Коренът поддържа всички каталози и файлове. В присъщата на Linux файлова йерархия отсъства концепцията „устройство”, за разлика от тази на Windows. Вместо това файловите системи биват монтирани в каталог така, че се организира единна интегрирана дървовидна структура. Без значение е дали файловата система е разположена на локално или на отдалечено устройство. Файловата система интегрира единна йерархия на файловете, която започва с корена „/”.Достъпът до елементите (файлове и каталози) в йерархичната структура се задава чрез „пътечка”, която в абсолютното си представяне започва с корена „/”.

Достъпът до пълната файлова йерархия в Android е възможен само при наличие на необходима оторизация на привилегирован потребител “root”, ползващ се със „системни” администраторски права. „Нормалните” потребителски правомощия за ползване на файловата система са ограничени до безопасно равнище и предоставят достъп само до част от файловата йерархия.

Ядрото на OS Android, на което се базира OS Android, разполага с възможности да поддържа работата с много широк диапазон от файлови системи – от Journal File System (JFS) за AIX (вариант на IBM за Unix) до файловата система Amiga (…). Всички операции във файловата система се извършват с посредничеството на абстрактен слой от ядрото, наречен Virtual File System (VFS). Всяка файлова система притежава собствен модул в ядрото, който регистрира поддържаните от нея операции във VFS. Чрез разделянето на реализацията на операциите от абстрактното им представяне, добавянето на нова файлова система е просто въпрос на включване на нов модул към ядрото на OS Android. По принцип тези модули или са част от ядрото на OS, или се добавят динамично при необходимост. По този начин целият процес при монтиране на файловата система е скрит от потребителя. Обикновено ядрото на OS Android съдържа модулите само на тези файлови системи, които имат отношение към изпълняваните от нея операции.

Въпреки, че Android е базирана на ядрото на Linux, йерархията на нейната файлова система не съответства изцяло на стандарта Filesystem Hierarchy Standart [4], който определя характеристиките на файловите системи, основаващи се на Linux. Android използва няколко раздела за да организира каталозите и файловете на устройството. Всеки от тези раздели има различна роля във функционалността на устройството, въпреки че обикновеният потребител (а често и начинаещите разработчици) не си дава сметка за значението и за съдържанието на тези раздели. Стандартните раздели на вътрешната памет на Android-устройствата са: /boot, **/system**, **/recovery**, **/data**, **/cache** и **/misc**. В допълнение към основните раздели, следва да бъдат отбелязани и тези от SD-картата: **/sdcard** и **/sd-ext**. Тези раздели играят съществена роля във функционирането на OS Android.

Съществено е да е отбележи ролята на три от разделите, които не се срещат в стандартните Linux базирани системи: **/system**, **/data** и **/cache.** В процеса на изграждане (генериране) на операционната система се създават три файла-образи (image files) – **system.img**, **userdata.img** и **cashe.img**, които имат основни функции в работата на системата. При първоначалното зареждане на ОС програмата init монтира тези файлове на строго определени места във файловата система – съответно това са посочените три раздела - **/system**, **/data** и **/cache**.

Разделът **/system** обобщава състава на цялата операционна система Android с изключение на ядрото на Linux, което е разположено изцяло в раздела **/boot**. Този раздел съдържа подкаталозите **/system/bin** и **/system/lib**, които съдържат изпълними файлове на ядрото в процесорно-зависим код. Наред с това разделът обхваща и всички системни приложения, които се създават с генерирането на системния образ. Директорията се монтира като достъпна само за четене (read-only) и съдържанието ѝ не може да бъде променяно по време на изпълнение.

Тъй, като съдържанието на раздела **/system** е недостъпно за промяна, за съхраняване на данните, свързани с потребителските приложения се използва разделът **/data**. Така например директорията **/data** съдържа всички **.apk** файлове на инсталираните приложения, докато поддиректорията **/data/data** включва "home" директориите на приложенията.

Разделът **/cache** е отговорен за съхраняването на данни и компоненти на приложения, до които често се осъществява достъп. Тук се съхраняват и някои от получените актуализации на операционната система преди да бъдат изпълнени.

Директориите **/system**, **/data** и **/cache** се формират в процеса на компилация на операционната система, а правата за достъп по подазбиране и идентификаторите на собствениците на файлове и каталози, съдържащи се в съответните файлове-образи се получават от конфигурационните файлове, съдържащи списък от системно предефинирани идентификатори на потребители и групи.

Допълнителните раздели **/sdcard** и **/sd-ext** по принцип са свързани с използването на външна SD-карта, а не на вътрешната памет на устройството. На устройства, които притежават както вътрешна, така и външна SD-карта, разделът **/sdcard** винаги реферира вътрешната такава. За реферииране на външната SD-карта (когато присъства) се използва алтернативен раздел с име, което се различава за различните устройства. Обикновено пространството в тези раздели се ползва за потребителски данни или за логическо разширение на раздела **/data**.

5. **Основни принципи на сигурността в OS Android**.

Най-важният принцип в подхода на сигурността, приет в Android е да не се допусне случайно или злонамерено увреждане на системни ресурси, както и на потребителски приложения и данни. Като начално, базово средство за постигане на някаква степен на сигурност се изплзва разделянето на постоянната памет на устройството на два дяла: за съхранение на системните компоненти и за съхранение на данни (вкл.потребителски). Системният дял се монтира като достъпен само за четене (read-only), с цел да се избегне манипулирането на жизненоважните компоненти на операционната система. „Потребителският" дял е мястото, където се запазват приложенията и поддържаните постоянни данни в системата. В някои случаи при инсталирането на нови приложения режимът на достъп до системния дял може да се промени чрез повторно монтиране на системния дял, с оглед регистрирането на необходимата информация в системата.

Най-общият подход към сигурността в Android използва многослойната организация на операционната система, като механизмът на сигурността се реализира в две равнища: на равнището на ядрото на Linux и на равнището на слоя Application Framework. В първия случай всяко приложение се изпълнява в специфична „пясъчна кутия" (Application Sandbox). Ядрото налага изолиране на компонентите на приложенията и на операционната система, като използва стандартните средства на Linux (разделяне и изолиране на процесите и механизмът,

известен като Discretionary Access Control – DAC – „избирателно управление на достъпа”). Изолацията се постига чрез отбелязването на всяко прилжение с индивидуален идентификатор, образуван от идентификаторите на Unix-потребителя (UID) и групата (GID). Тези идентификатори се присвояват в процеса на инсталиране на приложението и не се променят до неговото деинсталиране. По този начин всяко приложение в Android се асоциира с кореспондиращ Unix-потребител. Така обозначеното приложение се изпълнява в собствен Linux-процес, напълно изолиран от работата на другите процеси в системата, с ограничен достъп до функциите и услугите на операционата система (в съответствие с реда за изпълнение на процесите в Android, отбелязан по-горе).

Механизмът DAC е типичен за реализиране на сигурността на файловете в рамките на файловата система на Linux (Android). Достъпът до файла се определя от създателя или от собственика на ресурса и третира правата на три група потребители – самият собственик на файла, потребителите от същата група и всички останали потребители („останалия свят”). За всеки тип потребители се установяват група от права за достъп във вид „четене-запис-изпълнение” (r-w-x).

Описаният DAC-модел, характерен за Linux позволява ограничен контрол на достъпа на елементарно, твърде примитивно равнище. В много от случаите той може сравнително лесно да бъде зобиколен чрез осъществяване на индиректен достъп до целевите файлови ресурси.

Съществена концепция в сигурността на Android са разрешенията (permissions). Следва да се обърне внимание, че те са напълно различни от правата за достъп до файловете в Linux. Системата от разрешения в Android, която по същество е представена от средната фаза в модела на сигурността (фиг. ...), следва подхода Mandatory Access Control (MAC – мандатно/принудително управление на достъпа) и е насочена към едно значително по-прецизно диференцирано и по-детайлно описание на правата на достъп. Разрешенията са обявени под формата на предефинирани символни низове и се използват широко за управление на достъпа на компонентите между различните приложения. Всички разрешения се заявяват и получават по време на инсталиране на приложенията. За да бъде получено разрешението, то следва да бъде изискано в манифестния файл на приложението (**AndroifManifest.xml**) където се специфицират атрибутите и характеристиките на приложенията по определена синтактична форма,. Операционната система проверява заявените разрешения и взема окончателно решение дали да ги предостави. При стартирането на приложението се предизвиква текуща проверка на предоставените му разрешения, преди да бъде получен реален достъп до изискваните от него ресурси. Така например, едно приложение (напр.игра) никога няма да получи достъп до Internet, ако липсва разрешение за такава връзка.

Операционната система предоставя на разработчиците над 60 различни вътрешно дефинирани разрешения. Те са дефинирани в пакета **`android.Manifest.permission.XXX`**, където XXX замества името на конкретното разрешение. В допълнение към вградените разрешения, разработчиците могат да дефинират и свои собствени разрешения – т.нар. „динамични разрешения” в Android. Дефинирането на такива разрешения става в манифестния файл на приложението и тяхната задача е да назначат желания ред за достъп до приложенията и техните компоненти. Изисква се имената на дефинираните разришения да са уникални в глобален план и да са описателни така, че другите компоненти да са в състояние да могат ги разпознаят и да ги заявят по тяхното име.

Операционната система Android определя четири нива на защита при използването на системата от разрешения. Всяко такова ниво представлява параметър на разрешението и е необходимо да бъде специфицирано в случаите, когато се дефинира собствено разрешение. Всяко ниво на защита налага различна политика на сигурност. Градацията на нивата по посока на увеличаване на степента на защита изглежда така: „нормално” (normal), „опасно” (dangerous), „подпис” (signature) и „подпис или системна” (SignatureOrSystem).

Разрешенията от ниво „нормално” получават тази стойност по подразбиране и предоставят най-слаба защита. Те най-често се използват по отношение на функционалностите,

които са най-малко критични за сигурността на системата. Ако нивото на защита не е посочено, то се приема, че предоставената стойност е “normal”. Когато приложението при инсталирането си изисква „нормална” защита, то тя се предоставя без за това да се уведоми потребителя. Във всеки такъв случай потребителят все пак има възможност да провери всички предоставени разрешения на инсталираното от него приложение.

Разрешенията от ниво „опасно” се предоставят по усмотрение на потребителя на устройството. Тези разрешения могат да се отнасят до личните данни на потребителя или до различни функции на хардуера. Възможно е предоставянето на разрешения от тази група да активират функционалности, водещи до икономически вреди за потребителя на устройството. Когато приложението изисква предоставяне на разрешения от ниво dangerous, системата уведомява за това потребителя, като му предоставя въможност да вземе решение относно инсталирането му. Отказът от страна на потребителя от приемането на което и да е от исканите разрешения от ниво dangerous води до прекратяване на процеса на инсталация.

Разрешенията от групата signature се проверяват от операционната система, която взема решение за предоставянето им, без участието на потребителя. За да бъде дадено такова разрешение, изисикващото го приложение трябва да бъде подписано със същия ключ както приложението, което е защитено чрез това разрешение. Условието да бъде разрешен достъпът между две приложения на ниво на защита signature е всяко от тях да бъде подписано с един и същи ключ.

Разрешенията от ниво SignatureOrSystem се характеризират от две условия, като удовлетоворяването на което и да е от тях дава право на желания достъп до ресурса. Едното от условията изисква пакетът да бъде разположен в системния образ, докато другото изисква пакетът, който се изисква достъпа, да бъде подписан със същия ключ както пакета, разположен в сиситемния образ.

Google използва статичен подход по отношение механизма на разрешенията. В Android всички разрешения се изискват и се получават по време на инсталация на приложенията. След като бъдат одобрени, те повече не могат да бъдат променяни и остават валидни до деинсталирането на прилжението. Получените разрешения могат да бъдат проверявани от системата преди всеки актуален опит за достъп или при изрично изискване от разработчика.

6, Процес на първоначално зареждане на Android

Последователността от действия по първоначално зареждане на Android в някои случаи може да бъде различна за различните видове мобилни устройства, но следва да се отележи, че след зареждането на Linux-ядрото в RAM, процесът протича по напълно стандартен начин, независимо от вида и модела на устройството.

До първоначалното включване на захранването, устройството е в неинициализирано състояние. При включване на захранването процесорът започва да изпълнява поредица от команди, стартирайки от предефниран хардуерен адрес. Този адрес сочи към част от кода в памет, достъпна само за четене (write-protected memory), в която е локализиран т.нар.Boot ROM. Главната задача на кода, разположен в Boot ROM е да се определи средата, където се намира модулът Boot Loader. Когато тази среда бъде идентифицирана, Boot ROM зарежда Boot Loader във вътрешната памет (достъпна след включване на устройството), след което се изпълнява преход към зареждащия код на Boot Loader. Boot Loader е малка програма, която се изпълнява преди стартирането на Android и не е част от операционната система. Стартирането на Boot Loader установява външната RAM-памет, файловата система, и поддръжката на мрежовите операции. След изпълнението на тези действия се зарежда в паметта Linux-ядото, на което се предава управлението. Linux-ядрото инициализира средата за изпълнение на C-код, активира контролерите за обработкана прекъсванията, установява управлението на паметта, инициализира диспечирането на процесите, зарежда драйверите и монтира корена на файловата система. След инициализацията на модулите за управление, ситемата е в

състояние да използва виртуалната памет и да изпълнява процеси в потребителското пространство.

Първият стартиран процес в потребителското пространство е процесът init. Изпълнимият код на тази програма е разположен в главната директория на файловата система (root). Процесът init е „прародител" на всички други процеси в Android. Той е отговорен за създаването на всички базисни входни елементи на файловата система (създаване на директории и монтиране на устройства). След изпълнението на тези операции, процесът init анализира съдържанието на конфигурационния файл (скрипт) init.rc и изпълнява командите, записани в него.

Конфигурационният файл **init.rc** съдържа списък от действия, чието изпълнение се стартира от предефинирани събития. Командите, записани в конфигурационния файл init.rc дефинират системни глобални променливи, установяват основни параметри на ядрото за управление на паметта, конфигурират файловата система и т.н. От съществено значение за сигурността на системата е, че **init.rc** е отговорен за създаване на структурата на базисната файлова система и за присвояването на атрибут „собственик" и за установяването на права на достъп във файловата система за създадената структура.

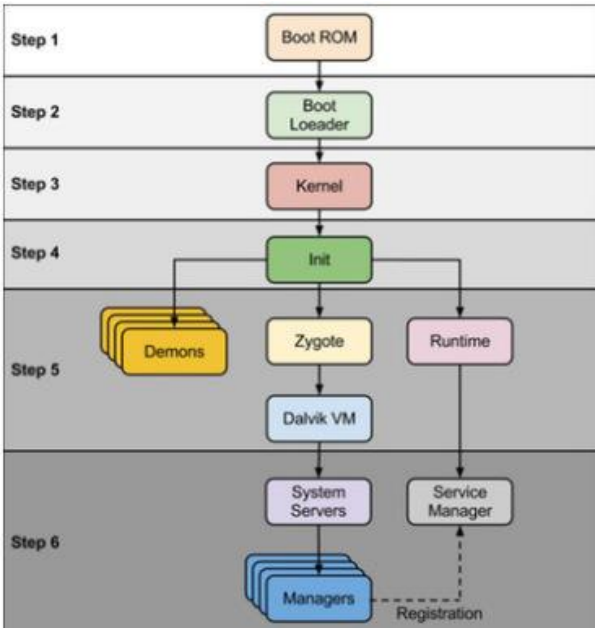
Допълнително програмата init е отговорна и за стартирането на няколко съществени фонови процеса (демони) и други процеси в Android, чиито параметри също са дефинирани във файла **init.rc**. Всеки изпълняван процес в Linux по подразбиране използва същите права на достъп (има същия UID), както неговия прародител. В Android init се стартира с root-привилегии (с UID = 0). Това означава, че всички породени от него процеси ще работят със същия UID. Всеки привилегирован процес е в състояние да промени UID на породените от него процеси към такива с по ниски привилегии, което се използва при създаването на процеси-потомци на init.

Една от основните услуги, които се стартира от програмата init е service manager, която служи като като индекс за всички други услуги, изпълнявани в Android. Тя трябва да бъде достъпна на един по-ранен етап, защото всички системни услуги, които се стартират след това, трябва да имат възможността да се регистрират и така да станат видими за другите компоненти на операционната система.

Друг важен процес, който се стартира от процеса init е Zygote. Това е специален процес, който е прародител на всички процеси (в които се изпълняват приложения на Java - DVM). Когато се стартира ново приложение, Zygote се саморазклонява, след което всички параметри, свързани с новото приложение (UID, GID и др.) се установяват за дъщерния процес. При стартирането на новия процес се достига съществено ускорение, тъй като в неговото пространство не се създава копие на основните библиотеки, необходими за изпълнението му. Паметта на новия процес има защита от типа "copy-on-write", което означава, че даните ще се копират от процеса Zygote в новия процес, ако последният се опитва да пише в защитената памет. По този начин основните библиотеки не могат да бъдат променени – те се разполагат само на едно място в паметта, намалявайки разхода на памет и времто за стартиране на приложението.

Първият процес, който се изпълнява с изплзването на Zygote е System Server. Този процес най-напред стартира услугите Surface Fliger и Sensor Service. След инициализирането на тези услуги, се изпълняват методи на System Server, които стартират останалите услуги (). Всички стартирани услуги се регистрират в ServiceManager.

Със стартирането на System Server процесът на първоначално зареждане на OS Android завършва. Краят на този процес се обозначава от системно оповестяващо съобщение, наречено **`ACTION_BOOT_COMPLETED`**. За да се стартира собствена услуга или да се изпълнят някакви действия, активирани със завършването на процеса на първоначално зареждане, е необходимо те да се регистрират чрез Alarm Manger така, че да получат това оповестяващо съобщение.



Фиг. ... .Схема на изпълнение на процеса на първоначално зареждане на OS Android.

Заключение.

Операционната система Android успя за кратко време да се наложи като доминираща операционна система на пазара на мобилните усройства. Нейните предимства произтичат от успешно приложения подход от съчетаване на проверени архитектурни решения, отворен код, достъпност и възможностите за покриване на постоянно разширяващи се разнообразни устройства и платформи, и съпътстващите ги хардуерни разширения (сензори, камери, средства за локализация и др). Извън всяко съмнение е, че тенденциите, отбелязани при създаването и развитието на тази операционна система ще оставят значима следа в разработването на специализиран системен софтуер за масови мобилни компютърно базирани усройства.

Литература:

1. Conder Shane, Lauren Darcey, Android™ Wireless Application Development Second Edition, Addison-Wesley, 2012.
2. Phillips Bill, Brian Hardy, Android Programming: The Big Nerd Ranch Guide, Big Nerd Ranch, Inc., 2013.
3. Dornin Laird, G. Blake Meike, Zigurd Mednieks and Masumi Nakamura, Programming Android O'Reilly, 2012.
4. Filesystem Hierarchy Standard, LSB Workgroup, The Linux Foundation, Version3.0.