BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC VĂN HIẾN

TÌM HIỂU HỆ ĐIỀU HÀNH ANDROID

TIỂU LUẬN MÔN HỌC HỆ ĐIỀU HÀNH

GVHD: Hồ Văn Ngọc

SVTH: Bùi Minh Nhật

MSSV: 201A290002

Buổi: Thứ sáu, ca 1

TP. HỒ CHÍ MINH, 2021

Mục lục

Chương I

Tổng quan về Android

1.1 Giới thiệu

Android™ là một hệ điều hành di động dựa trên nhân Linux (Linux kernel) được phát triển bởi Google. Nó được thiết kế chủ yếu cho các thiết bị màn hình cảm ứng như điện thoại thông minh và máy tính bảng. Phiên bản đầu tiên của Android được phát hành vào ngày 23 tháng 9 năm 2008, còn bản phát hành mới nhất là Android 11 vào ngày 8 tháng 9 năm 2020 [2].

Hình 1.1.1: Logo của hệ điều hành Android.[1]

Hệ điều hành Android được sử dụng thường xuyên nhất trên các nền tảng di động khác nhau trên thế giới. Nó chiếm khoảng 72% thị phần toàn thế giới trong năm qua [3]. Vào tháng 9 năm 2008, thiết bị Android đầu tiên được tung ra thị trường để thống trị ngành công nghiệp di động nhờ một số tính năng như thân thiện với người dùng, sự hỗ trợ của cộng đồng rất lớn, khả năng tùy biến, sản xuất thiết bị Android ở các công ty lớn. Do đó, thị trường xuất hiện nhu cầu phát triển các thiết bị hỗ trợ Android cùng với các nhà phát triển thông minh. Hệ điều hành Android đã trở thành một bộ hệ điều hành hoàn chỉnh cho các thiết bị khác nhau như thiết bị đeo được, điện thoại di động, máy tính xách tay, TV thông minh, máy tính bảng, hộp giải mã tín hiệu,...

[1]Nguồn: https://developer.android.com/distribute/marketing-tools/brand-guidelines.

Mã nguồn mở đã được sử dụng để phát triển các biến thể của Android trên một loạt các thiết bị điện tử khác, chẳng hạn như bảng điều khiển trò chơi, máy ảnh kỹ thuật số, trình phát đa phương tiện di động, PC và các loại khác, mỗi loại đều có giao diện người dùng chuyên biệt. Một số sản phẩm phái sinh nổi tiếng bao gồm Android TV cho TV và Wear OS cho thiết bị đeo, đều do Google phát triển. Các gói phần mềm trên Android, sử dụng định dạng APK, thường được phân phối thông qua các cửa hàng ứng dụng độc quyền như Google Play Store, Samsung Galaxy Store, Huawei AppGallery, Cafe Bazaar và GetJar hoặc các nền tảng mã nguồn mở như Aptoide hoặc F-Droid.

1.2 Lịch sử

Lịch sử của Android bắt đầu từ tháng 10 năm 2003 - trước khi thuật ngữ điện thoại thông minh được sử dụng rộng rãi, và vài năm trước khi Apple công bố iPhone và iOS đầu tiên của mình. Android Inc được thành lập tại Palo Alto, California. Bốn người sáng lập của nó là Rich Miner, Nick Sears, Chris White và Andy Rubin. Rubin đã tiết lộ trong một bài phát biểu năm 2013 tại Tokyo rằng hệ điều hành Android ban đầu được dùng để cải thiện hệ điều hành của máy ảnh kỹ thuật số [9]. Nhưng ngay cả khi đó, thị trường máy ảnh kỹ thuật số đang suy giảm. Chỉ vài tháng sau, Android Inc quyết định chuyển hướng sang sử dụng hệ điều hành bên trong điện thoại di động.

Hình 1.2.2: T-Mobile G1/HTC Dream – Thiết bị thương mại đầu tiên chạy Android.[2]

Trở lại năm 2005, lúc mọi người đều nghĩ về Google chỉ là một công ty tìm

[2]Nguồn: https://manual-user-guide.com/htc-dream.

kiếm hỗ trợ quảng cáo. Tuy nhiên, gần 16 năm trước, vào ngày 11 tháng 7 năm 2005, Google đã mua lại một công ty khởi nghiệp nhỏ có tên là Android. Google đã dành ba năm tiếp theo để phát triển hệ điều hành cho thiết bị di động. Điều này lên đến đỉnh điểm khi ra mắt phiên bản Android công khai đầu tiên vào năm 2008, được phát hành trên T-Mobile G1/HTC Dream.

1.3 Ưu điểm và nhược điểm của Android

1.3.1 Ưu điểm

- Android thuộc sở hữu của Google, một trong những tổ chức đáng tin cậy và có uy tín nhất trên toàn cầu. Android được cung cấp dưới giấy phép mã nguồn mở và hoàn toàn miễn phí.
- Android hoạt động trên hầu hết các thiết bị và linh hoạt.
- Có những widget cho phép thực hiện nhanh chóng công việc.
- Android cho phép chạy đa nhiệm. Chúng ta có thể xử lí nhiều tác vụ cùng một lúc.
- Google một trung tâm ứng dụng khổng lồ Google Play, nơi ta có thể tìm thấy hàng triệu ứng dụng cho thiết bị Android của mình. Android có nhiều ứng dụng hơn bất kỳ nền tảng hệ điều hành nào khác.
- Android có nhiều tính năng so với các hệ điều hành khác; đồng thời cũng cho phép tùy biến và chỉnh sửa nhiều hơn.

1.3.2 Nhược điểm

- Quảng cáo Android thường xuyên xuất hiện nhất trên các ứng dụng Android phổ biến miễn phí, điều này gây khó chịu cho người dùng.
- Hệ điều hành Android được đánh giá là một trong những hệ điều hành ngốn pin nhất. Trong hệ điều hành Android, có rất nhiều quá trình chạy ngầm dẫn đến việc hao pin nhanh chóng.
- Thường thì các ứng dụng Android đi kèm với bảo mật thấp.

- Nếu RAM của điện thoại di động ít thì rất dễ gặp tình trạng treo máy, kể cả khi chỉ mở vài tác vụ.
- Android cần nhiều mã nguồn hơn (so với IOS).
- Ứng dụng có chứa vi-rút còn tồn tại trong Android Market.

Chương II

Kiến trúc Android

Hình ảnh ở trang kế bên cho cái nhìn tổng quát về kiến trúc Android.

2.1 Máy ảo Dalvik

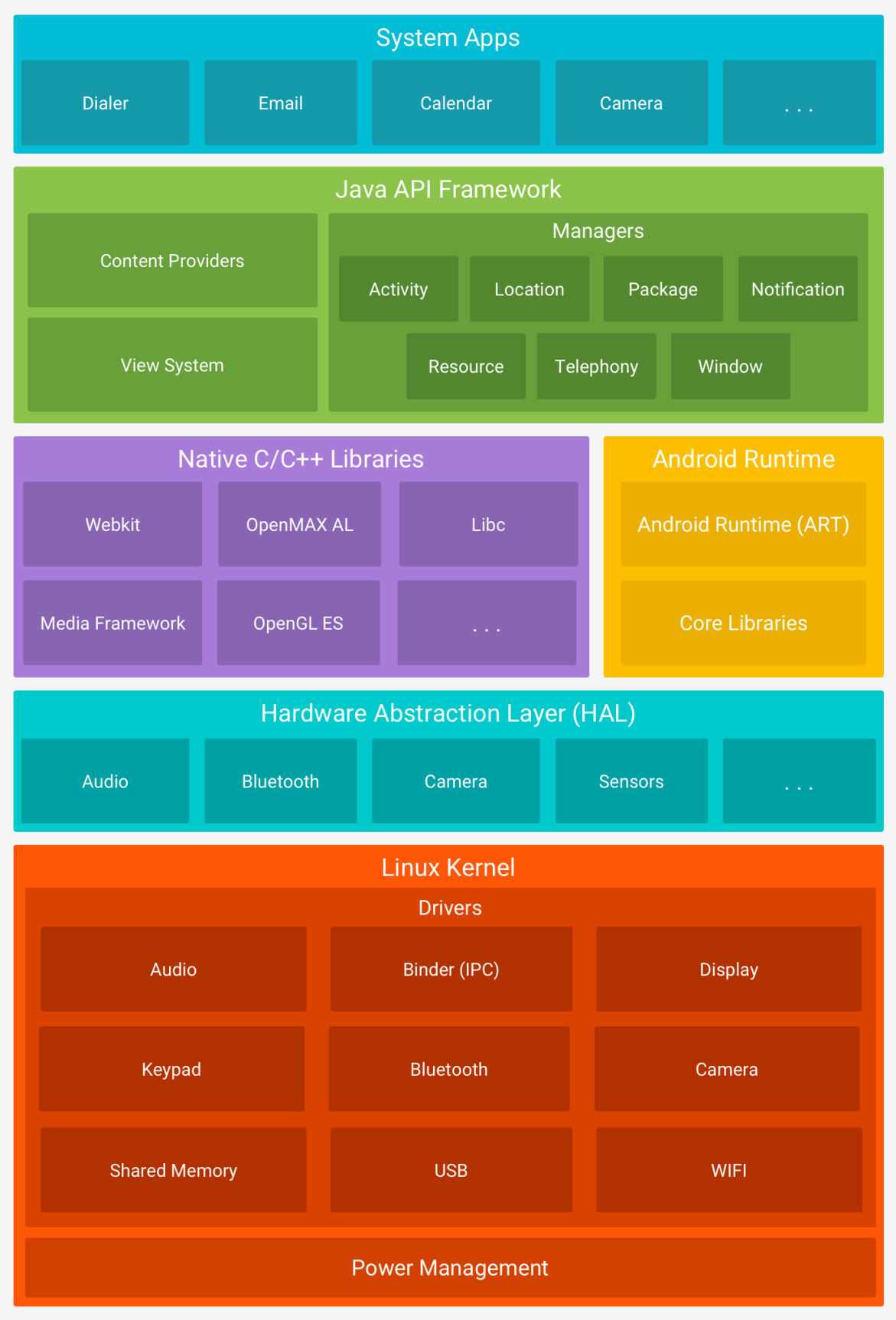
Dalvik là máy ảo giúp các ứng dụng java chạy được trên các thiết bị động Android. Nó chạy các ứng dụng đã được chuyển đổi thành một file thực thi (.dex) để phù hợp cho các hệ thống hay bị hạn chế về bộ nhớ và tốc độ xử lý.

Hệ thống Android thực thi máy ảo Dalvik sử dụng byte-code đặc biệt, thay vì thực thi trực tiếp byte-code Java thuần. Cộng đồng Android cung cấp một công cụ (dx) cho phép chuyển đổi các tệp class Java thành tệp thực thi Dalvik (.dex). Việc triển khai máy ảo Dalvik được tối ưu hóa cao để hoạt động hiệu quả nhất có thể trên các thiết bị di động thường được trang bị một CPU (đơn) khá chậm, tài nguyên bộ nhớ hạn chế, không có không gian hoán đổi hệ điều hành, và dung lượng pin hạn chế. Máy ảo Dalvik đã được triển khai theo cách cho phép một thiết bị thực thi nhiều máy ảo một cách khá hiệu quả.

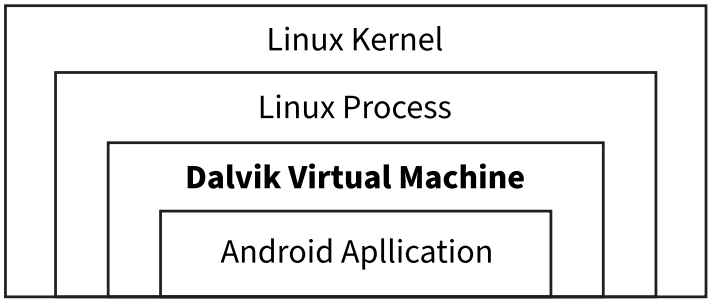
Cũng cần phải chỉ ra rằng máy ảo Dalvik dựa trên nhân Linux, cung cấp bất kỳ chức năng quản lý bộ nhớ cấp thấp và phân luồng nào. Trong Dalvik có bốn loại bộ nhớ khác nhau để phân biệt, có thể được nhóm lại thành bộ nhớ sạch/bẩn và chia sẻ/riêng tư. Dữ liệu điển hình nằm trong bộ nhớ sạch được chia sẻ hoặc riêng tư là các thư viện và các tệp ứng dụng cụ thể như tệp .dex. Bộ nhớ sạch được sao lưu bởi các tệp hoặc các nguồn khác và có thể được kernel "tỉa bớt" mà không làm mất dữ liệu. Nhóm phát triển Android nhận thấy rằng định dạng tệp .dex đã cắt giảm một nửa kích thước của một số thư viện hệ thống phổ biến và ứng dụng đi kèm với Android.

Với Android 2.2, một số thay đổi lớn đối với cơ sở hạ tầng của máy ảo Java đã được thực hiện. Với việc phát hành Android 2.2, trình biên dịch tức thời (just-in-time,

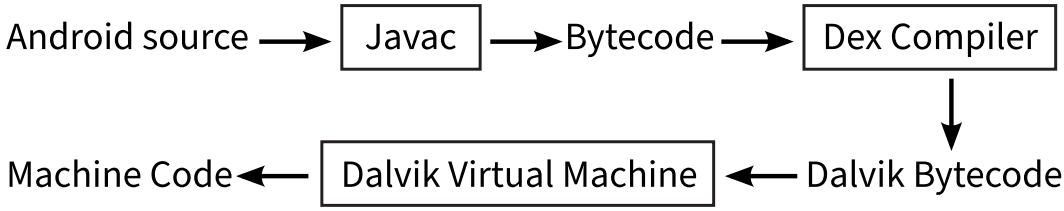
[1]Nguồn: https://developer.android.com/guide/platform/.



Hình 2.0.1: Tổng quát về kiến trúc Android theo tầng.[1]



Hình 2.1.2: Vị trí của máy ảo Dalvik trong Android



Hình 2.1.3: Sơ đồ quy trình dịch mã Android thành mã máy

JIT) đã được tích hợp vào, giúp dịch Dalvik byte-code thành mã máy hiệu quả hơn nhiều (tương tự như trình biên dịch C). Về cơ bản, các tính năng JIT, và thu gom rác (garbage collection) bổ sung sẽ được triển khai với Android, giúp tăng cường hiệu suất hệ thống tổng hợp hơn nữa.

2.1.1 Zygote

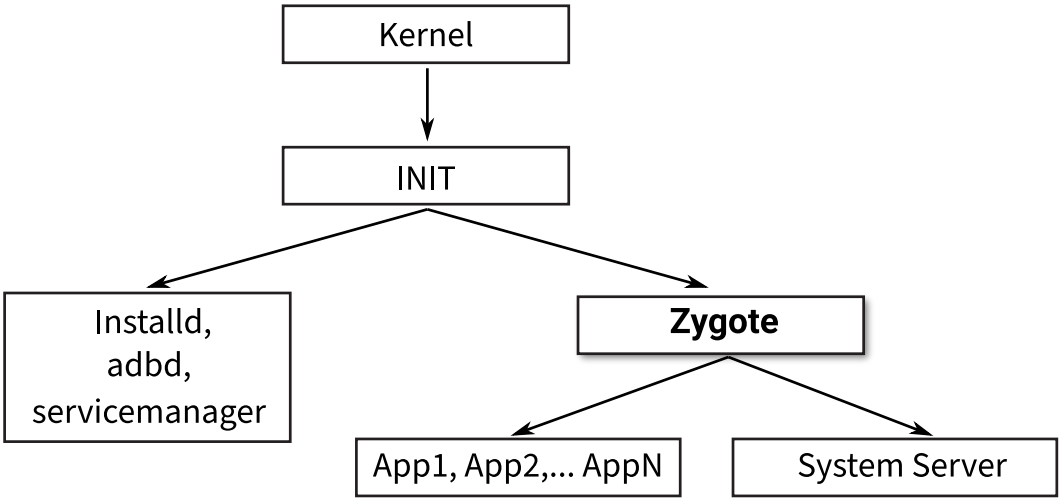
Vì mọi ứng dụng đều chạy trong phiên bản máy ảo của chính nó, các phiên bản máy ảo phải khởi động nhanh chóng khi một ứng dụng mới được khởi chạy và dung lượng bộ nhớ của máy ảo phải ở mức tối thiểu. Android sử dụng một khái niệm gọi là Zygote để cho phép chia sẻ mã giữa các phiên bản máy ảo và cung cấp thời gian khởi động nhanh các phiên bản máy ảo mới.

Thiết kế Zygote giả định rằng có một số lượng đáng kể các class thư viện lõi và cấu trúc heap tương ứng được sử dụng trên nhiều ứng dụng. Nó cũng giả định rằng các cấu trúc heap này thường ở chế độ chỉ đọc. Nói cách khác, hầu hết các ứng dụng sử dụng dữ liệu và các class này, nhưng không bao giờ sửa đổi. Những đặc điểm này được khai thác để tối ưu hóa việc chia sẻ bộ nhớ qua các tiến trình.

Zygote là một tiến trình máy ảo bắt đầu vào thời gian khởi động hệ thống. Khi Zygote bắt đầu, nó khởi tạo một máy ảo Dalvik, máy này được load trước và khởi tạo trước các class thư viện lõi. Nói chung, các class thư viện cốt lõi này ở trạng thái chỉ

đọc, do đó phù hợp để load trước và chia sẻ qua các tiến trình. Khi Zygote đã được khởi tạo, nó sẽ đợi các yêu cầu socket đến từ tiến trình thời gian chạy cho biết rằng nó sẽ phân nhánh các phiên bản máy ảo mới dựa trên phiên bản máy ảo Zygote. Bằng cách tạo ra các quy trình máy ảo mới từ Zygote, thời gian khởi động được rút ngắn.

Các class thư viện cốt lõi được chia sẻ trên các phiên bản máy ảo thường ở quyền chỉ được đọc bởi các ứng dụng. Khi các class đó được ghi vào, bộ nhớ từ tiến trình Zygote chia sẻ sẽ được sao chép vào tiến trình con của máy ảo của ứng dụng và được ghi vào đó. Thao tác này cho phép chia sẻ bộ nhớ tối đa trong khi vẫn cấm các ứng dụng can thiệp vào nhau [1].



Hình 2.1.4: Zygote trong quá trình boot của Android

2.1.2 Kiến trúc dựa thanh ghi

Thông thường, những nhà phát triển triển khai máy ảo ưa chuộng các kiến trúc dựa trên ngăn xếp hơn các kiến trúc dựa trên thanh ghi. Sự ưa thích này chủ yếu là do sự đơn giản của việc triển khai máy ảo, dễ dàng viết một trình biên dịch back-end (hầu hết các máy ảo ban đầu được thiết kế để lưu trữ một ngôn ngữ duy nhất và mật độ mã (nghĩa là các tệp thực thi cho kiến trúc ngăn xếp luôn nhỏ hơn các tệp thực thi cho kiến trúc thanh ghi)" [6].

Sự đơn giản và mật độ mã đi kèm với chi phí hiệu suất. Các nghiên cứu đã chỉ ra rằng kiến trúc dựa trên thanh ghi yêu cầu và trung bình các lệnh máy ảo được thực thi ít hơn 47% so với kiến trúc dựa trên ngăn xếp. Mặt khác, mã thanh ghi nhiều hơn 25% so với mã ngăn xếp tương ứng, nhưng chỉ tải thêm máy thực 1,07% cho mỗi lệnh

máy ảo, là không đáng kể. Hiệu suất tổng thể của máy ảo dựa trên thanh ghi là trung bình mất ít hơn 32,3% thời gian để thực thi các benchmark tiêu chuẩn [7].

2.2 Nền tảng đích - tập lệnh

Để đơn giản, ở phần sau tiểu luận đề cập các thiết bị dựa trên Linux 2.6 là hệ thống x86, nhưng hầu hết điện thoại di động là sản phẩm dựa trên ARM. Trong khi ARM đại diện cho kiến trúc tập lệnh rút gọn (reduced instruction set computer, RISC) 32 bit, các hệ thống x86 chủ yếu dựa trên kiến trúc tập lệnh phức tạp (complicated instruction set computer, CISC). Nói chung, có thể khẳng định rằng ARM (RISC) đang thực hiện các lệnh đơn giản hơn (nhưng nhiều hơn) so với hệ thống x86 (CISC). Bộ nhớ là phần rất quan trong các thiết bị di động, do có ràng buộc về kích thước, giá cả. và năng lượng sử dụng.

ARM giải quyết những vấn đề này bằng cách cung cấp tập lệnh 16 bit thứ 2 (second 16-bit) có thể được xen kẽ với các lệnh ARM 32 bit thông thường. Tập lệnh bổ sung này có thể giảm kích thước mã lên đến 30% (với một số hạn chế về hiệu suất). So với bộ vi xử lý x86, thiết kế ARM tập trung mạnh vào việc tiêu thụ điện năng thấp hơn, điều này một lần nữa làm cho nó phù hợp với các thiết bị di động [8].

2.3 Kernel và tiến trình khởi động

2.3.1 Kernel

Việc dựa trên Linux kernel cho phép Android tận dụng các tính năng bảo mật chính. Mặc dù Android dựa trên Linux 2.6, nhưng Android không sử dụng nhân Linux tiêu chuẩn [5]. Do đó, thiết bị Android không nên được quy vào là Linux. Một số cải tiến cụ thể trên Linux kernel của Android bao gồm:

- Trình điều khiển báo thức (cung cấp bộ hẹn giờ để đánh thức thiết bị). Đây là phần triển khai kernel để hỗ trợ AlarmManager của Android. Nó cho phép user space thông báo cho kernel nào nó muốn thức dậy, cho phép kernel lên lịch thích hợp và quay lại khi thời gian đã hết hạn, bất kể trạng thái ngủ của CPU.
- Trình điều khiển bộ nhớ chia sẻ (ashmem). Hệ thống con ashmem là một trình cấp phát bộ nhớ dùng chung mới, tương tự như POSIX SHM nhưng có hành vi

khác và thể hiện một API dựa trên tệp (file-based) đơn giản hơn.

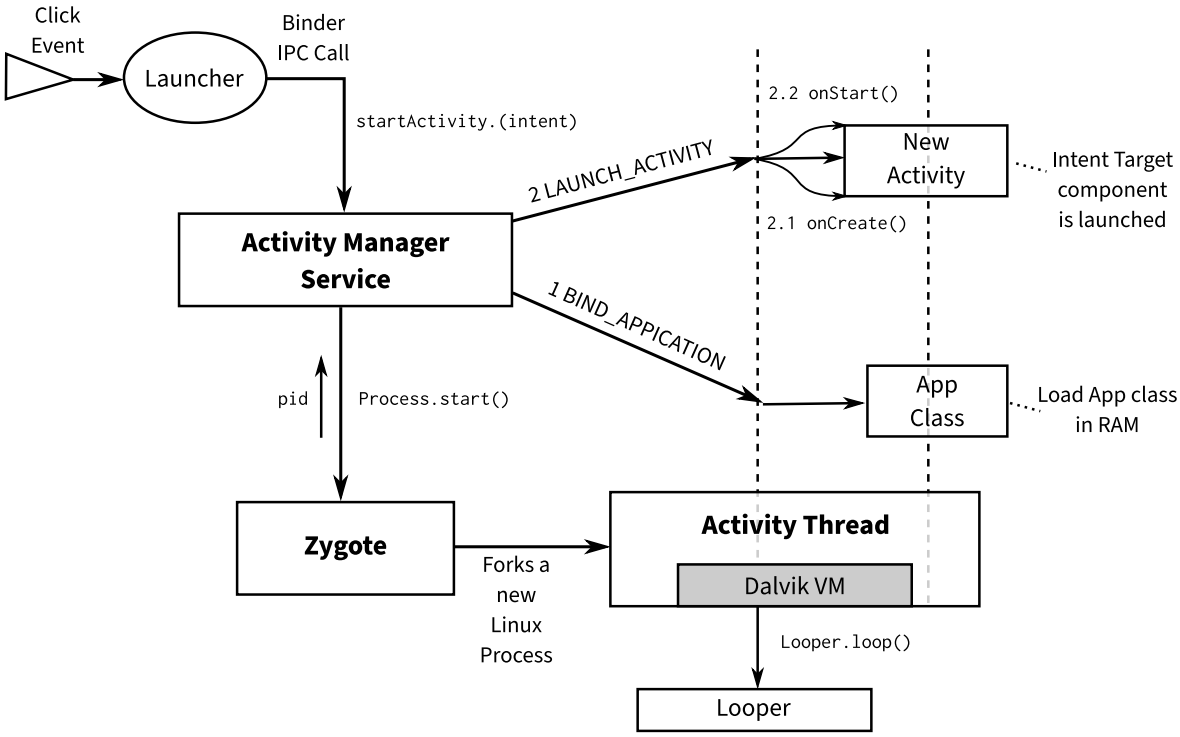
- "Chất kết dính" binder giao tiếp giữa các quá trình. Binder là một cơ chế giao tiếp liên quy trình dành riêng cho Android và hệ thống gọi thủ tục từ xa tương tự như DBus.
- Quản lý nguồn (có cách tiếp cận tích cực hơn so với Linux PM).
- Hạn chế tình trạng low memory (oom handling). Được biết đến với cái tên Viking Killer, OOM handler chỉ đơn giản ngắt các tiến trình mỗi khi bộ nhớ khả dụng trở nên thấp.
- Trình gỡ lỗi kernel và logger.

2.3.2 Tiến trình khởi động

Trong tiến trình khởi động Android, thành phần kernel Android-Linux trước tiên gọi quá trình init (giống Linux tiêu chuẩn). Quá trình init truy cập vào các tệp `init.rc` và `init.device.rc` (`init.device.rc` là thiết bị cụ thể). Ngoài tệp `init.rc`, quá trình Zygote được bắt đầu. Quy trình Zygote load các class Java cốt lõi và thực hiện các bước xử lí khởi tạo ban đầu. Các class Java này có thể được sử dụng lại bởi các ứng dụng Android, do đó, bước này xúc tiến quá trình khởi động tổng thể. Sau quá trình load ban đầu, Zygote chạy chậm lại trên socket và chờ các yêu cầu khác.

Mỗi ứng dụng Android chạy trong một quy trình riêng biệt, có máy ảo Dalvik riêng. Android là hệ điều hành người dùng đơn (single user OS), các nhà thiết kế chỉ định cho mỗi ứng dụng một UID duy nhất tại thời điểm cài đặt. Điều này có nghĩa là kernel Linux bên dưới có thể bảo vệ từng tệp và bộ nhớ ứng dụng. Các ứng dụng Android nhìn ở bậc thấp nhất là các quy trình Linux. Mỗi ứng dụng chạy như một tiến trình riêng và theo mặc định có 1 luồng.

Mọi ứng dụng Android đều chạy trong môi trường quy trình riêng của nó. Binder cho phép thông tin liên lạc bên trong các quá trình (inter-process communications, IPC) hiệu quả. Các đối tượng thực tế được lưu trữ trong bộ nhớ chia sẻ. Bằng cách sử dụng bộ nhớ dùng chung, IPC được tối ưu hóa, vì phải truyền ít dữ liệu hơn. So với hầu hết các môi trường Linux hoặc UNIX, Android không cung cấp bất kỳ không gian hoán đổi (swap space) nào. Do đó, dung lượng bộ nhớ ảo bị chi phối bởi dung lượng bộ nhớ vật lý có sẵn trên thiết bị [4].



Hình 2.3.5: Sơ đồ khởi động ứng dụng của Android

2.4 Thư viện Bionic

Google đã phát triển một thư viện tùy chỉnh cho trình biên dịch C (libc) được gọi là Bionic. Điều này là cần thiết vì ba lý do chính:

Giấy phép: Họ muốn giữ giấy phép GP ngoài user-space. Mã Bionic sử dụng giấy phép BSD.

Kích thước: Thư viện phải được tải trong mỗi tiến trình, vì vậy nó cần phải nhỏ. Bionic có kích thước khoảng 200KB, đâu đó chỉ bằng một nửa kích thước của glibc (phiên bản GNU của libc).

Tốc độ: Sức mạnh CPU hạn chế yêu cầu thư viện cần phải nhanh. Bionic có kích thước nhỏ và đường dẫn mã nhanh, bao gồm việc thực thi rất nhanh và nhẹ các pthread tùy chỉnh.

Bionic có hỗ trợ tích hợp cho các dịch vụ quan trọng dành riêng cho Android như thuộc tính hệ thống và ghi nhật ký. Nó không hỗ trợ một số tính năng POSIX nhất định không cần thiết trên Android, như ngoại lệ C ++. Do đó, nó không hoàn toàn

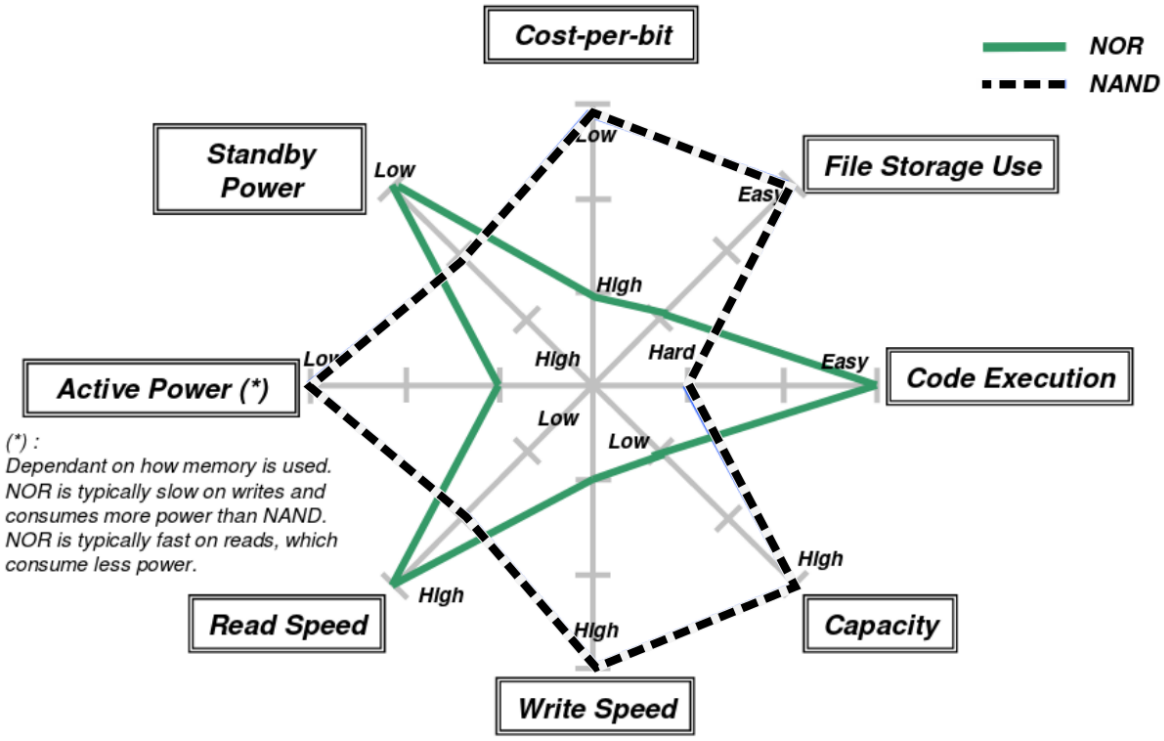
tương thích với GNU libc. Tất cả mã gốc phải được biên dịch dựa trên Bionic, không phải glibc.

Bảo mật được duy trì bằng cách gán một cặp ID người dùng (UID) và ID nhóm (GID) duy nhất cho mỗi ứng dụng. Vì các thiết bị di động thường chỉ được sử dụng bởi một người dùng (so với hầu hết các hệ thống Linux), cài đặt UNIX/Linux `/etc/passwd` và `/etc/group` đã bị xóa. Ngoài ra để tăng cường bảo mật, `/etc/services` đã được thay thế bằng một danh sách các dịch vụ (được duy trì bên trong chính tệp thực thi). Tóm lại, thư viện Android C đặc biệt thích hợp để hoạt động trong điều kiện CPU và bộ nhớ hạn chế, đây là trường hợp phổ biến đối với các nền tảng Android di động. Hơn nữa, các điều khoản bảo mật đặc biệt đã được thiết kế và thực hiện để đảm bảo tính toàn vẹn của hệ thống.

2.5 Phương tiện lưu trữ và File hệ thống

Khi nói đến cấu hình và thiết lập thiết bị di động, sử dụng ổ cứng truyền thống nói chung thì kích thước quá lớn, và tiêu thụ quá nhiều điện năng. Ngược lại, các thiết bị bộ nhớ flash thường cung cấp khả năng truy cập đọc tương đối nhanh cũng như khả năng chống sốc động học tốt hơn so với ổ cứng. Về cơ bản, có hai loại thiết bị bộ nhớ flash phổ biến, một dựa trên NAND và một dựa trên NOR [10]. Mặc dù nói chung, các giải pháp dựa trên NOR cung cấp mật độ thấp, chúng có đặc điểm là các thành phần ghi chậm và đọc nhanh. Mặt khác, các giải pháp dựa trên NAND cung cấp chi phí thấp, mật độ cao và được xem là các giải pháp IO ghi nhanh và đọc chậm. Một số hệ thống nhúng đang sử dụng thiết bị flash NAND để lưu trữ dữ liệu và các thành phần dựa trên NOR cho môi trường thực thi mã.

Độ tin cậy của dữ liệu lưu là một khía cạnh quan trọng đối với bất kỳ thiết bị nhớ nào. Bộ nhớ flash bị một hiện tượng gọi là đảo bit, trong đó một số bit có thể bị đảo ngược. Hiện tượng này phổ biến hơn trong NAND flash hơn là NOR Flash. NAND flash được vận hành với các bad block nằm rải rác ngẫu nhiên khắp nơi. Nhiều ô nhớ bị hỏng sau các chu trình xóa và tiếp tục, trong suốt vòng đời của NAND flash. Do đó, xử lý bad block là một yêu cầu bắt buộc đối với NAND flash. Mặt khác, NOR flash được vận hành không có bad block nào với sự tích lũy bad block rất thấp trong suốt thời gian hoạt động của bộ nhớ. Do đó, khi nói đến độ tin cậy của dữ liệu được lưu trữ, NOR flash có lợi thế hơn NAND flash.



Hình 2.5.6: So sánh flash NAND và NOR.[2]

Từ góc độ file hệ thống, kể từ phiên bản Android 2.3, file hệ thống Linux ext4 nổi tiếng được sử dụng. Trước file hệ thống tệp ext4, Android thường sử dụng một hệ thống flash khác là YAFFS. YAFFS được biết đến là file flash hệ thống Linux đầu tiên được tối ưu hóa NAND. Một số nhà cung cấp sản phẩm Android (chẳng hạn như Archos với ext3 trong Android 2.2) đã thay thế file hệ thống Archos tiêu chuẩn bằng một file hệ thống khác mà họ lựa chọn. Cách đây chừng một thập kỉ, kích thước tối đa của bất kỳ ứng dụng Android nào đều tính bằng megabyte, so với các hệ thống dựa trên Linux, được coi là rất nhỏ. Điều này gợi ý cho ta rằng các yêu cầu về bộ nhớ và file hệ thống từ góc độ kích thước, các thiết bị chạy Android rất khác biệt so với hầu hết các hệ thống Linux.

2.6 Quản lí điện năng (pin)

Trong lĩnh vực thiết bị di động, quản lý điện năng rõ ràng là điều cực kỳ quan trọng. Quản lý điện năng trong bất kỳ hệ thống công nghệ thông tin nào, với bất kỳ hệ điều hành nào, được coi là một nhu cầu cần thiết, do nhu cầu tiêu thụ điện ngày càng

[2]Nguồn: [10].

tăng của các hệ thống máy tính ngày nay. Các hệ thống dựa Linux cung cấp các tính năng tiết kiệm năng lượng như định mức xung nhịp, chia tỷ lệ điện áp, kích hoạt chế độ ngủ, hoặc tắt bộ nhớ đệm. Mỗi tính năng này đều làm giảm mức tiêu thụ điện năng của hệ thống, thường là yêu cầu tăng độ trễ lên. Hầu hết các hệ thống dựa trên Linux quản lý mức tiêu thụ điện năng thông qua Cấu hình nâng cao và Giao diện nguồn (Advanced Configuration and Power Interface, ACPI).

Các hệ thống dựa Android cung cấp cơ sở hạ tầng quản lý điện năng của riêng chúng (Power Manager) được thiết kế dựa trên tiền đề rằng bộ xử lý không được tiêu thụ bất kỳ năng lượng nào nếu không có ứng dụng hoặc dịch vụ nào thực sự yêu cầu điện năng. Android yêu cầu các ứng dụng và dịch vụ yêu cầu tài nguyên CPU thông qua khóa chế độ thức thông qua khung ứng dụng Android và các thư viện Linux bản địa. Nếu không có khóa chế độ đánh thức (wake locks) nào hoạt động, Android sẽ tắt bộ xử lý.

Android 9

Android 9 giới thiệu một tính năng quản lý pin mới là App Standby Buckets. Nó giúp hệ thống ưu tiên các yêu cầu tài nguyên của ứng dụng, dựa trên mức độ mở gần đây và tần suất được sử dụng. Dựa trên các pattern tìm được bằng machine learning, mỗi ứng dụng được đặt vào một trong 5 kiểu ưu tiên. Hệ thống giới hạn tài nguyên thiết bị có sẵn cho từng ứng dụng dựa trên nhóm ứng dụng đó. Những hạn chế này chỉ áp dụng khi thiết bị đang sử dụng pin; hệ thống không áp đặt hạn chế này đối với các ứng dụng trong khi thiết bị đang sạc.

Chương III

Thành phần chính của ứng dụng Android

Các ứng dụng Android được đóng gói thành một gói Android (.apk) thông qua Android Asset Packaging Tool (AAPT). Để hợp lý hóa quá trình phát triển, Google cung cấp Công cụ phát triển Android (Android Development Tools, ADT). ADT sắp xếp hợp lý việc chuyển đổi từ file class sang file dex và tạo .apk trong quá trình triển khai. Chương này tìm hiểu về thành phần chính trong các ứng dụng Android nói chung.

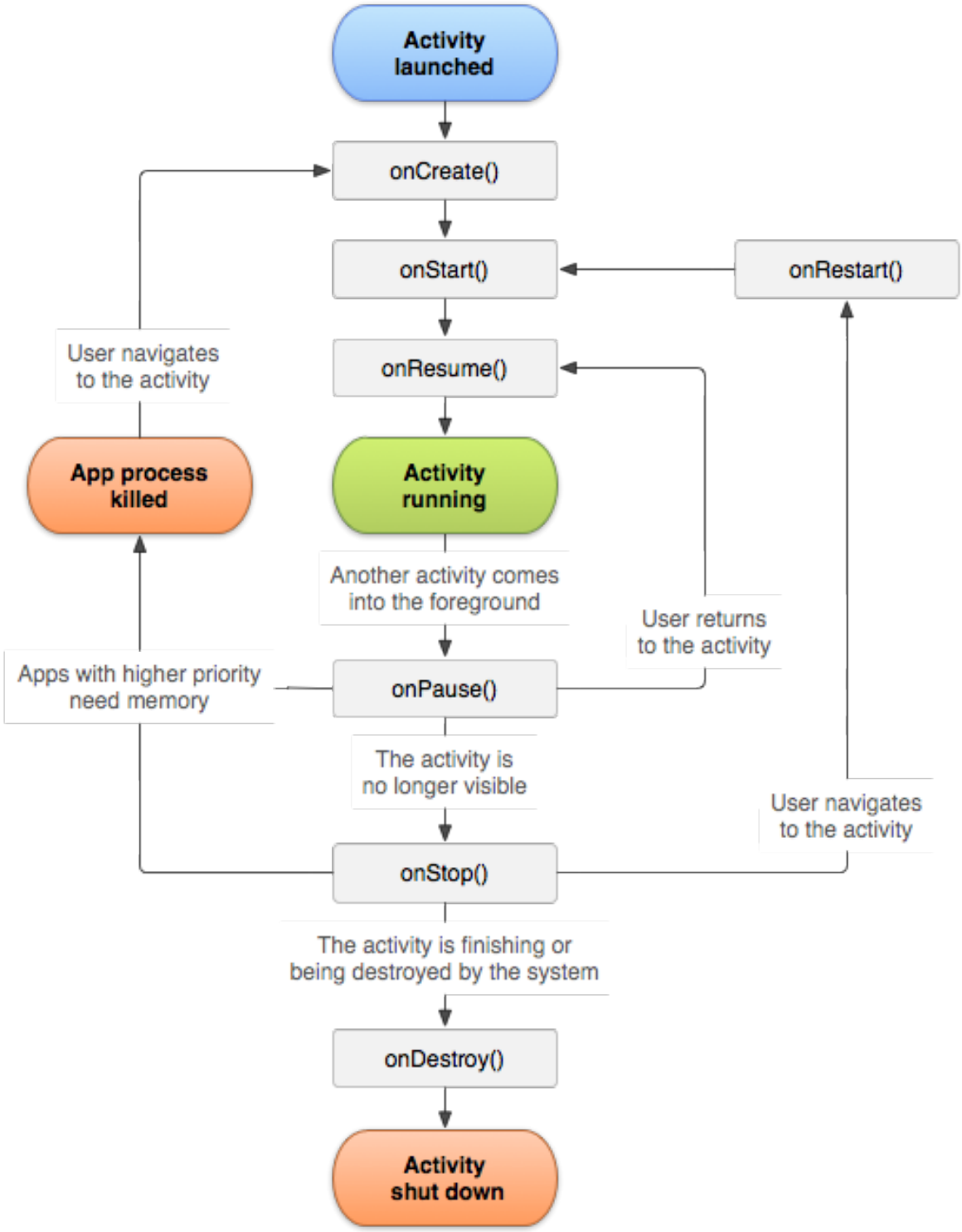
3.1 Activity

Activity là một thành phần quan trọng của ứng dụng Android, và cách các hoạt động được khởi chạy và kết hợp với nhau là một phần cơ bản của mô hình ứng dụng trên nền tảng này. Không giống như các mô hình lập trình trong đó các ứng dụng được khởi chạy bằng phương thức `main()`, hệ thống Android khởi tạo mã trong một cá thể Activity bằng cách gọi các phương thức callback cụ thể tương ứng với các giai đoạn cụ thể trong vòng đời của nó.

Trải nghiệm ứng dụng dành cho thiết bị di động khác với trên máy tính để bàn ở chỗ không phải lúc nào tương tác của người dùng với ứng dụng cũng bắt đầu ở cùng một nơi. Thay vào đó, hành vi của người dùng thường bắt đầu ở một nơi không xác định. Ví dụ, chúng ta vào YouTube trên ứng dụng trình duyệt Chrome, sau đó Chrome chuyển hướng ta thẳng đến ứng dụng Youtube, thay vì ta phải về màn hình menu để mở YouTube.

Class `Activity` được thiết kế để hỗ trợ mô hình này. Khi một ứng dụng gọi một ứng dụng khác, ứng dụng sẽ gọi một activity trong ứng dụng kia, thay vì toàn bộ ứng dụng. Theo cách này, activity đóng vai trò là điểm khởi đầu cho sự tương tác của ứng dụng với người dùng. Chúng ta triển khai một hoạt động như một subclass của class `Activity`.

[1]Nguồn: https://developer.android.com/guide/components/activities/activity-lifecycle/.



Hình 3.1.1: Sơ đồ vòng đời của Activity[1]

Trong các phương thức callback vòng đời, chúng ta có thể khai báo activity của mình hoạt động như thế nào khi người dùng rời đi và quay lại. Ví dụ, ta đang xây dựng trình phát video trực tuyến, ta có thể tạm dừng video và ngắt kết nối mạng khi người dùng chuyển sang ứng dụng khác. Khi người dùng quay lại, ta có thể kết nối lại với mạng và cho phép người dùng tiếp tục video từ thời điểm dừng lúc nãy. Nói cách khác, mỗi lệnh callback cho phép ta thực hiện công việc cụ thể phù hợp với một trạng thái thay đổi nhất định. Thực hiện đúng công việc vào đúng thời điểm và xử lý chuyển đổi đúng cách giúp ứng dụng của chúng ta hoạt động hiệu quả hơn. Việc triển khai tốt các lệnh gọi lại trong vòng đời có thể giúp đảm bảo rằng ứng dụng Android tránh được:

- Sự cố xảy ra nếu người dùng nhận được cuộc gọi điện thoại bất chợt, hoặc chuyển sang ứng dụng khác khi đang sử dụng ứng dụng của ta.
- Tốn tài nguyên hệ thống khi người dùng sử dụng ít tương tác với ứng dụng.
- Làm mất tiến trình của người dùng nếu họ rời khỏi ứng dụng và quay lại ứng dụng sau đó. Hoặc làm mất tiến trình của người dùng khi họ xoay màn hình.

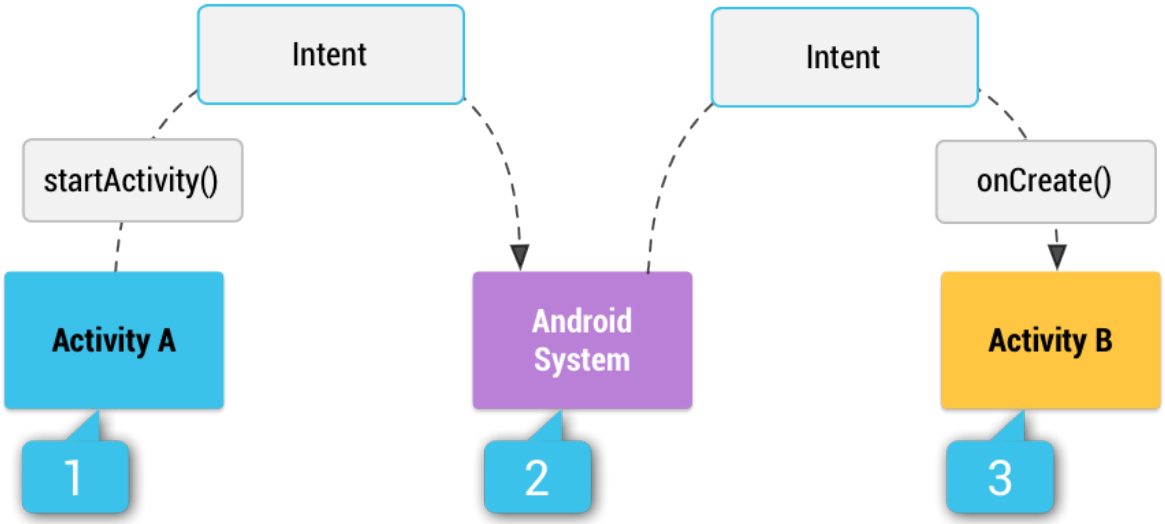
3.2 Intents

Intent là một đối tượng nhắn tin mà chúng ta có thể sử dụng để yêu cầu một hành động từ một thành phần ứng dụng khác. Đối tượng Intent mang thông tin mà hệ thống Android sử dụng để xác định thành phần nào sẽ bắt đầu, cùng với thông tin mà thành phần người nhận sử dụng để thực hiện đúng hành động (chẳng hạn như hành động cần thực hiện và dữ liệu để thực hiện). Mặc dù intent tạo điều kiện giao tiếp giữa các thành phần theo một số cách, nhưng có ba trường hợp sử dụng cơ bản:

Bắt đầu một activity: Activity đại diện cho một màn hình duy nhất trong một ứng dụng. Chúng ta có thể bắt đầu một phiên bản mới của Activity bằng cách passing Intent tới hàm `startActivity()`. Intent mô tả activity cần bắt đầu và mang theo tất cả dữ liệu cần thiết.

Bắt đầu một dịch vụ: Tương tự Activity, chúng ta có thể khởi động dịch vụ để thực hiện thao tác một lần (chẳng hạn như download file) bằng cách passing Intent tới `startService()`. Intent mô tả dịch vụ cần bắt đầu và mang theo tất cả dữ liệu cần thiết.

Cung cấp một chương trình phát sóng (broadcast): Broadcast là một thông báo mà bất kỳ ứng dụng nào cũng có thể nhận được. Hệ thống cung cấp các broadcast khác nhau cho các sự kiện của hệ thống, chẳng hạn như khi hệ thống khởi động hoặc khi thiết bị bắt đầu sạc. Chúng ta có thể gửi chương trình phát sóng đến các ứng dụng khác bằng cách passing Intent tới `sendBroadcast()` hoặc `sendOrderedBroadcast()`.



Hình 3.2.2: Cách intent được sử dụng khi bắt đầu một hoạt động. Khi Intent chỉ định rõ ràng một thành phần hoạt động cụ thể, hệ thống sẽ khởi động thành phần đó ngay lập tức. [2]

3.3 Dịch vụ

Dịch vụ là một thành phần thực hiện các hoạt động nền trong thời gian dài mà không có giao diện người dùng. Sau khi bắt đầu, một dịch vụ có thể liên tục chạy trong một thời gian, ngay cả sau khi người dùng chuyển sang ứng dụng khác. Ngoài ra, một thành phần có thể liên kết với một dịch vụ để tương tác với nó và thậm chí thực hiện giao tiếp giữa các quá trình. Ví dụ: một dịch vụ có thể xử lý mạng, phát nhạc, lướt web. Có ba loại dịch vụ khác nhau:

Foreground: Dịch vụ foreground thực hiện một số hoạt động mà người dùng dễ nhận thấy. Ví dụ: một ứng dụng âm thanh sẽ sử dụng dịch vụ foreground để phát một

[2]Nguồn: https://developer.android.com/guide/components/intents-filters/.

bản nhạc. Các dịch vụ nền trước phải hiển thị thông báo để người dùng chủ động biết rằng dịch vụ đang chạy.

Background: Dịch vụ nền thực hiện một hoạt động mà người dùng không trực tiếp nhận thấy. Ví dụ: logging.

Bound: Một dịch vụ bị ràng buộc (bound) khi một thành phần ứng dụng liên kết với nó bằng cách gọi `bindService()`. Dịch vụ ràng buộc cung cấp giao diện client-server cho phép các thành phần tương tác với dịch vụ, gửi yêu cầu, nhận kết quả. Loại dịch vụ này chỉ chạy khi nó liên kết với thành phần ứng dụng khác.

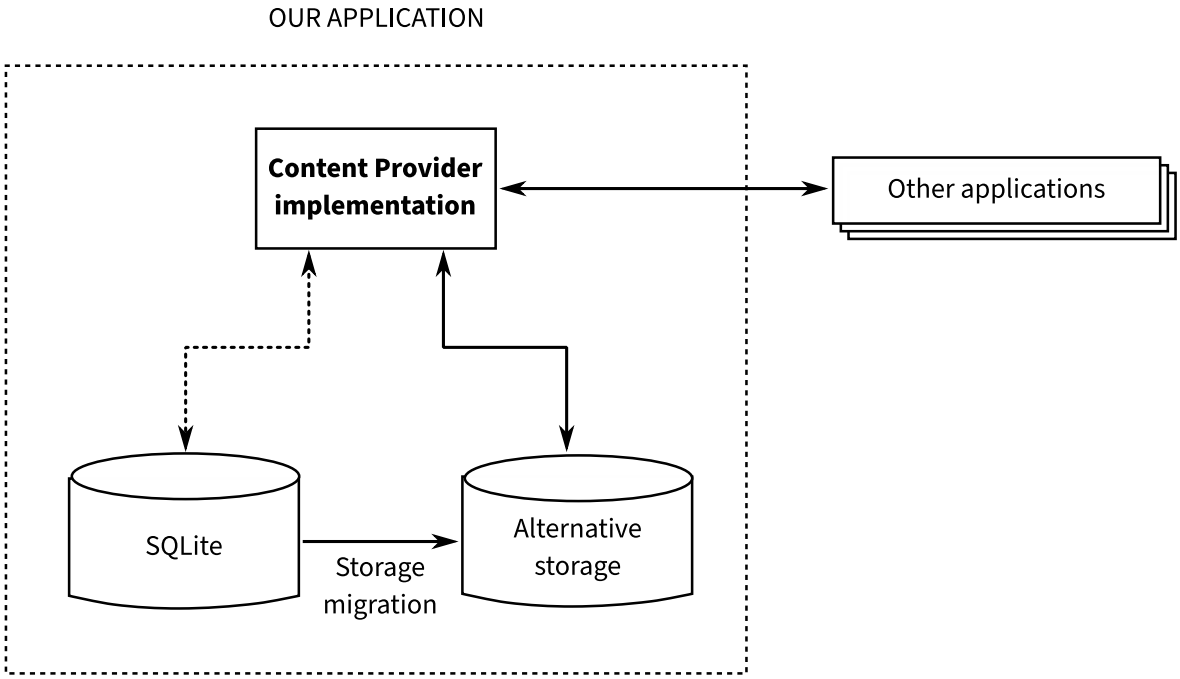
Bất kỳ thành phần ứng dụng nào cũng có thể sử dụng dịch vụ theo cách giống như cách mà bất kỳ thành phần nào cũng có thể sử dụng một activity — bằng cách khởi động nó với intent.

3.4 Nhà cung cấp nội dung

Nhà cung cấp nội dung (content provider) có thể giúp ứng dụng quản lý quyền truy cập vào dữ liệu do chính ứng dụng lưu trữ, được lưu trữ bởi các ứng dụng khác, và cung cấp cách chia sẻ dữ liệu với các ứng dụng khác. Chúng đóng gói dữ liệu, cung cấp các cơ chế để xác định bảo mật dữ liệu. Nhà cung cấp nội dung là interface tiêu chuẩn kết nối dữ liệu trong cùng một quy trình với mã chạy trong quy trình khác. Triển khai một nhà cung cấp nội dung có nhiều lợi thế. Quan trọng nhất là chúng ta có thể chỉ định cấu hình nhà cung cấp nội dung để cho phép các ứng dụng khác truy cập và sửa đổi dữ liệu ứng dụng của chúng ta một cách an toàn.

Các nhà cung cấp nội dung cung cấp quyền kiểm soát chi tiết đối với các quyền truy cập dữ liệu. Chúng ta có thể chọn hạn chế quyền truy cập vào một nhà cung cấp nội dung trong ứng dụng của mình, hoặc cấp quyền truy cập hàng loạt để truy cập dữ liệu từ các ứng dụng khác, hoặc định cấu hình các quyền khác nhau để đọc và ghi dữ liệu.

Ngoài ra ta cũng có thể sử dụng trình cung cấp nội dung để tóm tắt các chi tiết để truy cập các nguồn dữ liệu khác nhau trong ứng dụng của mình. Ví dụ: ứng dụng của chúng ta lưu trữ các bản ghi có cấu trúc trong cơ sở dữ liệu SQLite, cũng như các file ảnh và âm thanh. Chúng ta sử dụng nhà cung cấp nội dung để truy cập tất cả dữ liệu này.



Hình 3.4.3: Sơ đồ cách các nhà cung cấp nội dung quản lý quyền truy cập vào bộ nhớ.

Chương IV

Kết luận

Android là một hệ điều hành có sự phát triển rất ấn tượng. Sự thống trị của nó trong thị phần điện thoại thông minh rất rõ ràng. Nhưng vì quá lớn, Android khó tránh khỏi nhiều nhược điểm. Android sử dụng nhân Linux 2.6 đã được sửa đổi, tích hợp vào môi trường hệ điều hành của mình. Như đã thảo luận trong tiểu luận này, Android đã kế thừa, chỉnh sửa, tối ưu kernel cho các thiết bị có cấu hình thấp. Việc sử dụng mã nguồn mở cho phép nhiều nhà phát triển tham gia hơn, do đó hệ sinh thái Android ngày càng được mở rộng.

Tài liệu tham khảo

[1] Patrick Brady. Anatomy & physiology of an android, 2008 Google I/O Session Videos and Slides. ⟨https://sites.google.com/site/io/anatomy--physiology-of-an-android⟩.

[2] Dave Burke. Turning it up to android 11. ⟨https://blog.google/products/android/android-11/⟩.

[3] Statcounter GlobalStats. Mobile operating system market share worldwide june 2020 - june 2021. ⟨https://gs.statcounter.com/os-market-share/mobile/worldwide⟩.

[4] Dominique A. Heger. Quantifying it stability - 2nd edition. 2011.

[5] Sandra K. Johnson. Performance tuning for linux servers. IBM Press , 2005.

[6] Derek Jones. Register vs. stack based VMs. ⟨http://shape-of-code.coding-guidelines.com/2009/09/17/register-vs-stack-based-vms/⟩.

[7] Sohail Khan, Shahryar Khan, Syed Hammad Khalid Banuri, Mohammad Nauman, and Masoom Alam. Analysis of dalvik virtual machine and class path library. ⟨http://lim.univ-reunion.fr/staff/fred/Doc/Dalvik/Analysis-of-Dalvik-VM.pdf⟩.

[8] Frank Maker and Yu-Hsuan Chan. A survey on android vs . linux. 2009.

[9] Mitja Rutnik. Did you know: Android was originally designed for digital cameras not phones. ⟨https://www.androidauthority.com/android-history-digital-cameras-1111795/⟩.

[10] Toshiba. NAND vs. NOR flash memory: Technology overview. Toshiba , 2006.