AOSP

AOSP SOMMAIRE

Sommaire

1. **Comprendre ce qu'est AOSP**
    - Définition d'AOSP (Android Open Source Project)
    - Pourquoi AOSP est important pour le développement d'applications Android
2. **Comprendre l'architecture d'AOSP**
    - La structure des fichiers AOSP
    - Les différents éléments d'AOSP, tels que les noyaux Linux, les outils de compilation, les applications système, etc.
3. **Utiliser les outils AOSP**
    - Configuration de l'environnement de développement AOSP
    - Utilisation de l'outil repo pour récupérer les fichiers source AOSP
    - Compilation d'AOSP à partir des sources
4. **Comprendre le noyau Linux utilisé par AOSP**
    - Les fonctionnalités du noyau Linux utilisé par AOSP
    - Comment AOSP interagit avec le noyau Linux

Sommaire

5. **Comprendre les services Android**
    - Les différents services Android, tels que le gestionnaire de packages, le gestionnaire de processus, etc.
    - Comment les services Android interagissent avec les autres éléments d'AOSP
6. **Comprendre les applications système**
    - Les applications système les plus importantes, telles que l'interface utilisateur, le clavier, la messagerie, etc.
    - Comment les applications système interagissent avec les autres éléments d'AOSP
7. **Comprendre la gestion de la mémoire**
    - Comment AOSP gère la mémoire, y compris la mémoire virtuelle, la mémoire physique et la gestion de la mémoire pour les applications
8. **Comprendre la sécurité**
    - Les mécanismes de sécurité utilisés par AOSP, tels que SELinux, les autorisations d'application et la signature d'application

Sommaire

AOSP Partie 1

Comprendre ce qu'est AOSP

Définition d'AOSP (Android Open Source Project)

- L'AOSP, ou Android Open Source Project, est l'initiative de Google pour maintenir et gérer le code source libre et ouvert de la plateforme Android. Concrètement, il s'agit d'un ensemble de code source que tout le monde peut utiliser pour créer des versions personnalisées du système d'exploitation Android.
- Android est l'un des systèmes d'exploitation les plus populaires au monde pour les appareils mobiles comme les smartphones et les tablettes, et il peut également être utilisé sur des montres, des téléviseurs et d'autres appareils. Bien que beaucoup associent Android à Google, il est important de noter que Android, dans son essence, est un projet open source.

Comprendre ce qu'est AOSP

Pourquoi AOSP est important pour le développement d'applications Android

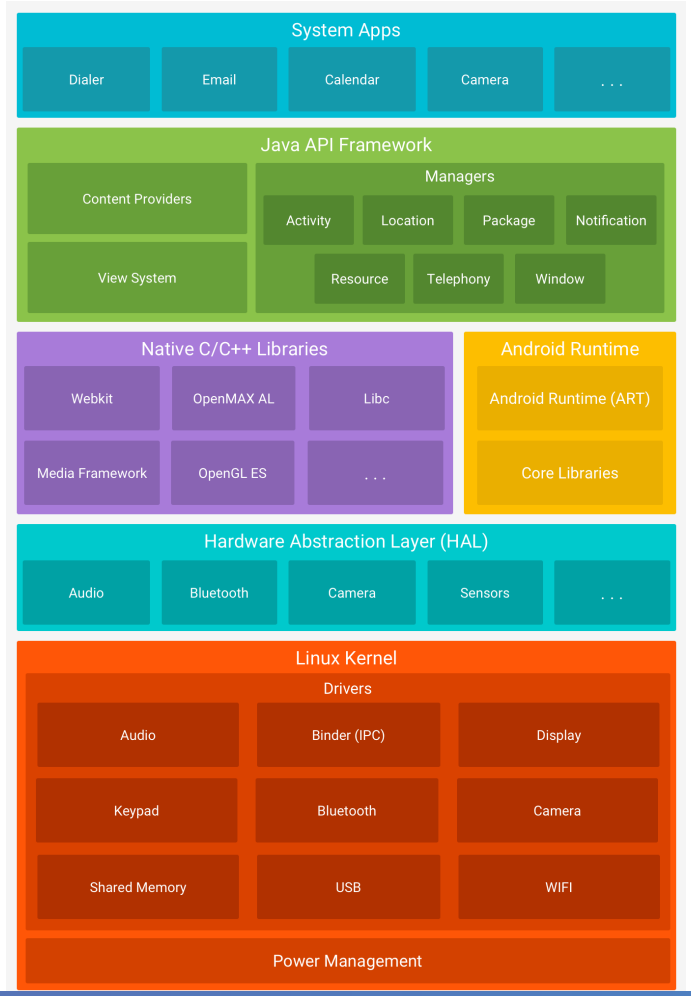
- **Accès au code source:** AOSP donne aux développeurs un accès direct au code source d'Android. Cela permet de comprendre en profondeur le fonctionnement interne du système d'exploitation, ce qui peut être crucial pour le développement d'applications avancées ou pour la résolution de problèmes complexes.
- **Personnalisation:** Les fabricants d'appareils ou les développeurs peuvent utiliser l'AOSP pour créer des versions personnalisées d'Android, adaptées à des besoins spécifiques. Par exemple, une entreprise peut développer une version d'Android spécifique pour ses appareils avec des fonctionnalités et des optimisations uniques.
- **Contributions à la communauté:** Étant donné que l'AOSP est open source, cela signifie que n'importe qui peut contribuer au projet. Si un développeur trouve un bug ou souhaite ajouter une nouvelle fonctionnalité, il peut le faire directement dans le code d'AOSP. De nombreuses améliorations et optimisations d'Android proviennent de la communauté des développeurs.

Comprendre ce qu'est AOSP

Pourquoi AOSP est important pour le développement d'applications Android

- **Écosystème diversifié:** L'AOSP a conduit à la création d'une multitude de versions d'Android, connues sous le nom de "ROMs personnalisées". Ces ROMs offrent souvent des fonctionnalités supplémentaires ou différentes optimisations par rapport à la version standard d'Android distribuée par Google. Cette diversité profite aux développeurs d'applications en élargissant l'éventail de configurations et de fonctionnalités avec lesquelles leurs applications peuvent interagir.
- **Indépendance vis-à-vis des services Google:** Les versions d'Android basées sur l'AOSP ne dépendent pas nécessairement des services Google. Cela signifie que les appareils utilisant ces versions peuvent fonctionner sans les applications et services Google préinstallés. C'est particulièrement important pour les régions ou les situations où les services Google ne sont pas disponibles ou souhaités.
- **Sécurité et confidentialité:** Les entreprises ou les individus préoccupés par la confidentialité ou la sécurité peuvent utiliser l'AOSP comme base pour construire leur propre version d'Android avec des améliorations ou des modifications spécifiques pour répondre à ces préoccupations.

Comprendre l'architecture d'AOSP



Comprendre l'architecture d'AOSP

- L'architecture d'AOSP est un ensemble complexe de logiciels qui interagissent pour fournir une expérience Android complète. Pour une compréhension claire, décomposons-le en composants essentiels et en structure de fichiers.

1. **Les différents éléments d'AOSP**

- **Noyau Linux:** C'est le cœur du système d'exploitation. Android utilise le noyau Linux pour ses fonctionnalités de bas niveau comme la gestion de la mémoire, la gestion des processus, le réseau, etc. Bien que basé sur le noyau Linux standard, le noyau Android contient souvent des modifications et des extensions spécifiques.
- **Bibliothèques natives:** Ce sont des bibliothèques écrites en C ou C++ qui sont compilées pour la plateforme Android. Elles fournissent des fonctionnalités clés pour le rendu graphique (comme Skia ou OpenGL ES), la base de données (SQLite) ou la médiation entre le noyau et le framework Android.
- **Runtime Android (ART):** ART (Android Runtime) est la machine virtuelle qui exécute le bytecode des applications Android. Il remplace l'ancienne machine virtuelle Dalvik.
- **Framework Android:** Il s'agit d'un ensemble de API Java sur lequel les applications Android sont construites. Cela comprend des services fondamentaux comme le gestionnaire de fenêtres, le gestionnaire de ressources, ou le gestionnaire d'activités.

Comprendre l'architecture d'AOSP

1. **Les différents éléments d'AOSP**

- **Outils de compilation:** Pour transformer le code source en une image système Android utilisable, plusieurs outils sont nécessaires. AOSP comprend des outils tels que AAPT (Android Asset Packaging Tool) pour emballer les ressources ou le compilateur DEX pour transformer le bytecode Java en format exécutable pour ART.
- **Applications et services système:** Ce sont des applications intégrées comme le lanceur, l'application téléphone, les SMS, les paramètres, etc. Elles sont généralement situées dans /packages/.
- **HAL (Hardware Abstraction Layer):** Il permet une normalisation des interfaces logicielles pour le matériel, rendant plus facile la création de firmware pour une variété de dispositifs sans avoir à réécrire des portions significatives du code système

les outils AOSP

1. **Configuration de l'environnement de développement AOSP**

- **Prérequis :**
  - Un ordinateur avec un système d'exploitation 64 bits, de préférence Linux (Ubuntu est souvent recommandé).
  - Au moins 250 Go d'espace libre sur votre disque dur.
  - 16 Go de RAM ou plus sont recommandés pour une compilation fluide.
- **Installation des dépendances :**
  - git, openjdk, ...

2. Utilisation de l'outil repo pour récupérer les fichiers source AOSP

- L'outil repo est un outil de gestion de dépôts Git spécialement conçu pour AOSP
  - Initialiser un dépôt
  - Télécharger les sources
  - Mises à jour

les outils AOSP

- L'Android Open Source Project (AOSP) est massif et comprend des dizaines (voire des centaines) de dépôts Git distincts. L'outil repo aide à gérer ces multiples dépôts comme s'il s'agissait d'un seul dépôt. Le manifeste du repo est le mécanisme qui permet de définir quels dépôts Git sont inclus dans un projet AOSP donné.
- Un manifeste personnalisé vous permet de spécifier vos propres versions ou branches de certains dépôts tout en utilisant les versions par défaut de l'AOSP pour d'autres.
- Demo

Comprendre l'architecture d'AOSP

2. **La structure des fichiers AOSP**

- Lorsque vous consultez le code source d'AOSP, vous rencontrerez une structure de répertoires spécifique. Voici une vue d'ensemble simplifiée:
  - **/bionic/:** Contient la bibliothèque C utilisée par Android, spécifiquement optimisée pour les appareils mobiles.
  - **/bootable/:** Code pour le bootloader et les images de récupération.
  - **/build/:** Scripts et configurations pour compiler AOSP.
  - **/cts/:** Android Compatibility Test Suite.
  - **/dalvik/:** L'ancienne machine virtuelle Android. Depuis Android 5.0, Dalvik a été remplacé par ART, mais des références à Dalvik peuvent encore être présentes dans les versions antérieures d'AOSP.
  - **/development/:** Outils et scripts pour aider au développement.

Comprendre l'architecture d'AOSP

2. **La structure des fichiers AOSP**
    - **/device/:** Configurations spécifiques au matériel pour différents appareils.
    - **/external/:** Bibliothèques tierces et autres composants open source.
    - **/frameworks/:** Le cœur des bibliothèques et des services Android.
    - **/hardware/:** Abstractions matérielles et interfaces pour le matériel Android.
    - **/packages/:** Applications et services Android, tels que le lanceur, le téléphone, etc.
    - **/prebuilts/:** Binaires préconstruits tels que les compilateurs.
    - **/system/:** Fichiers du système d'exploitation, y compris le noyau et les outils système.

Comprendre l'architecture d'AOSP

2. **La structure des fichiers AOSP**

- **Ajout d'applications système :**
  - /packages/apps/ : Pour ajouter de nouvelles applications système. Chaque application est généralement dans son propre sous-répertoire. Par exemple, l'application Horloge serait dans /packages/apps/Clock.
- **Ajout d'un nouveau périphérique :**
  - /device/ : Pour ajouter un nouveau périphérique, vous devrez créer un nouveau sous-répertoire ici spécifique à votre fabricant et modèle. Par exemple, /device/google/pixel pour un appareil Pixel de Google. Ce répertoire contiendra des configurations spécifiques à votre appareil, des scripts de démarrage, des définitions de partitionnement, etc.
  - /kernel/ : Si des modifications spécifiques au noyau pour votre appareil sont nécessaires, elles seraient généralement gérées ici, bien que la source du noyau puisse souvent être hébergée en dehors de l'arbre AOSP principal.
- **Modification du noyau :**
  - /kernel/ : Le noyau Android (basé sur Linux) peut avoir des modifications spécifiques pour Android. Si vous souhaitez apporter des modifications générales au noyau, c'est ici que cela se passe.

Comprendre l'architecture d'AOSP

2. **La structure des fichiers AOSP**

- **/external/** : Si vous intégrez des bibliothèques ou des outils open source tiers.
- **/hardware/** : Si vous développez ou modifiez une interface pour une nouvelle pièce matérielle ou un composant.
- **/development/** : Pour les outils et scripts liés au développement.
- **/bootable/** : Si vous travaillez sur le bootloader ou le mode de récupération.
- **Ce que vous ne devriez généralement pas toucher (à moins d'avoir une raison spécifique) :**
  - /bionic/ : La bibliothèque C d'Android. Sauf si vous savez ce que vous faites, les modifications ici peuvent avoir des répercussions majeures.
  - /system/ et /frameworks/ : Ces répertoires contiennent le cœur du système d'exploitation Android. Toute modification peut affecter la compatibilité et la stabilité de l'ensemble du système.

Comprendre l'architecture d'AOSP

2. **Quelques exemples d'éléments que nous pouvons ajouter**

- **/bootable/** :
  - **Objectif principal** : Contient le code pour les images bootables, notamment le bootloader et le mode de récupération.
  - **Ce que vous pouvez y faire** :
    - Personnaliser le mode de récupération (par exemple, en ajoutant des fonctions supplémentaires ou en modifiant son apparence).
    - Intégrer un nouveau bootloader ou modifier un bootloader existant pour un appareil spécifique. Ajouter des scripts ou des fonctions pour le processus de démarrage.

Comprendre l'architecture d'AOSP

- **/external/** :
  - **Objectif principal** : Bibliothèques et outils tiers open source.
  - **Ce que vous pouvez y faire** :
    - Ajouter une nouvelle bibliothèque open source qui pourrait être nécessaire pour votre appareil ou vos applications.
      Mettre à jour ou personnaliser les versions existantes des bibliothèques.
    - Corriger des bugs ou intégrer des patches à des bibliothèques existantes.
- **/hardware/** :
  - **Objectif principal** : Abstractions et interfaces pour le matériel.
  - **Ce que vous pouvez y faire** :
    - Développer une nouvelle interface HAL (Hardware Abstraction Layer) pour un composant matériel non encore pris en charge par Android.
    - Modifier une interface existante pour l'adapter aux besoins spécifiques d'un appareil.

Comprendre l'architecture d'AOSP

- **/development/** :
  - **Objectif principal** : Outils, exemples et scripts pour le développement.
  - **Ce que vous pouvez y faire** :
    - Ajouter de nouveaux outils ou scripts pour faciliter le développement ou le débogage.
    - Proposer des exemples d'applications ou de fonctionnalités pour aider les autres développeurs. Personnaliser ou étendre les outils de développement existants.
- **/packages/** :
  - **Objectif principal** : Applications et services du système Android.
  - **Ce que vous pouvez y faire** :
    - Ajouter une nouvelle application système ou un service.
    - Modifier ou personnaliser des applications existantes (par exemple, le lanceur, les paramètres, etc.).
    - Corriger des bugs ou ajouter des fonctionnalités à des applications existantes.

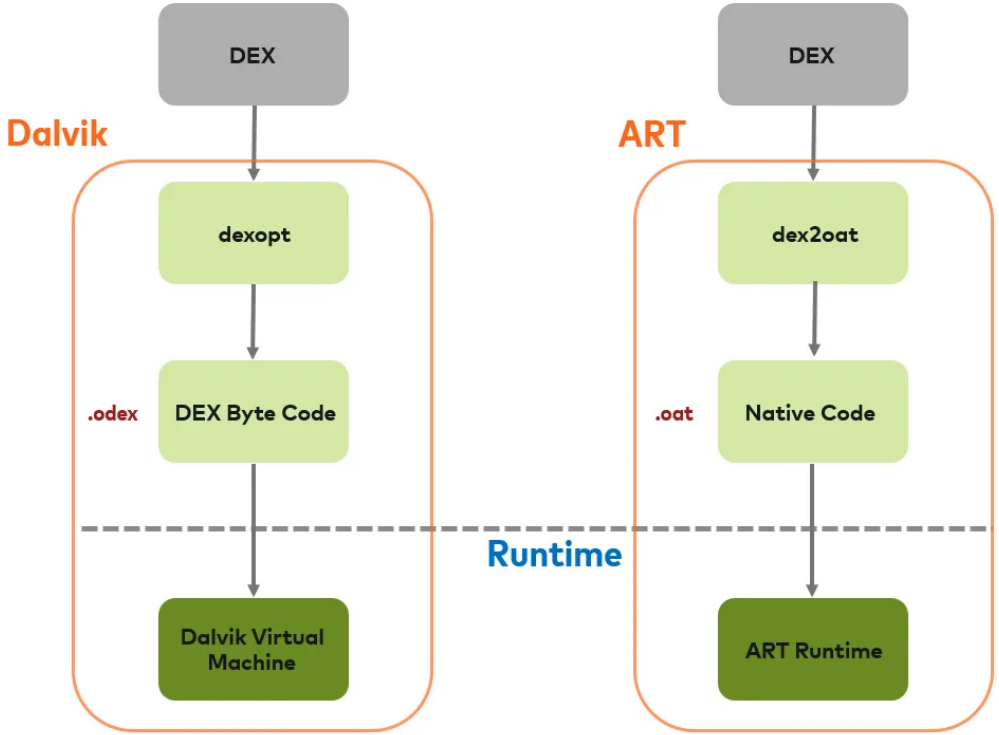
Comprendre l'architecture d'AOSP

- **/system/** :
  - **Objectif principal** : Fichiers du système d'exploitation, y compris des parties du noyau et des outils système.
  - **Ce que vous pouvez y faire** :
    - Intégrer des modifications ou des patches au noyau Android.
    - Ajouter ou modifier des outils système utilisés par Android.
- **/build/** :
  - **Objectif principal** : Scripts et configurations pour compiler AOSP.
  - **Ce que vous pouvez y faire** :
    - Personnaliser le processus de build, par exemple pour ajouter des étapes spécifiques ou modifier la configuration de build.

Comprendre l'architecture d'AOSP

1. **Runtime Android (ART)**
   ART a été introduit comme un remplacement de l'ancienne machine virtuelle Dalvik à partir d'Android 5.0 (Lollipop). ART exécute le bytecode dex, qui est une forme transformée du bytecode Java standard. Cette transformation est effectuée pour optimiser la taille et la performance du code sur les appareils Android. Les principales caractéristiques d'ART comprennent l'optimisation du code, l'amélioration des temps de réponse des applications, une meilleure gestion de la mémoire et une meilleure efficacité énergétique.

Comprendre l'architecture d'AOSP



Dalvik vs ART

Comprendre l'architecture d'AOSP

JIT (Just-In-Time Compilation) vs AOT (Ahead-Of-Time Compilation):

1. **JIT (Just-In-Time Compilation):**
   Fonctionnement: Avec JIT, le bytecode est compilé en code machine natif juste au moment où il est sur le point d'être exécuté. En d'autres termes, la compilation a lieu pendant l'exécution de l'application.
   Avantages:
   Optimisation au niveau de l'exécution : puisque le JIT sait exactement comment l'appareil fonctionne et quels sont les chemins d'exécution courants au moment de l'exécution, il peut effectuer des optimisations très spécifiques.
   Code universel: vous n'avez pas besoin de précompiler pour chaque architecture. Le bytecode reste le même, et le JIT s'occupe de la compilation spécifique à l'architecture.
   Inconvénients:
   Temps de démarrage : Il peut y avoir un délai la première fois qu'un code est exécuté car il doit être compilé.
   Consommation d'énergie : La compilation JIT consomme de l'énergie, ce qui peut être préjudiciable sur des appareils à batterie

Comprendre l'architecture d'AOSP

JIT (Just-In-Time Compilation) vs AOT (Ahead-Of-Time Compilation):

2. AOT (Ahead-Of-Time Compilation):

Fonctionnement: Avec AOT, le bytecode est compilé en code machine natif avant qu'il ne soit exécuté, c'est-à-dire lors de l'installation de l'application ou lors de la mise à jour du système.

Avantages:

Performance : Le code étant déjà compilé, l'application démarre rapidement et fonctionne efficacement dès le départ.

Moins de consommation d'énergie pendant l'exécution, car la compilation n'a pas lieu pendant que l'app est en cours d'utilisation.

Inconvénients:

Taille : La compilation AOT peut augmenter la taille de l'application car le code machine natif est généralement plus volumineux que le bytecode.

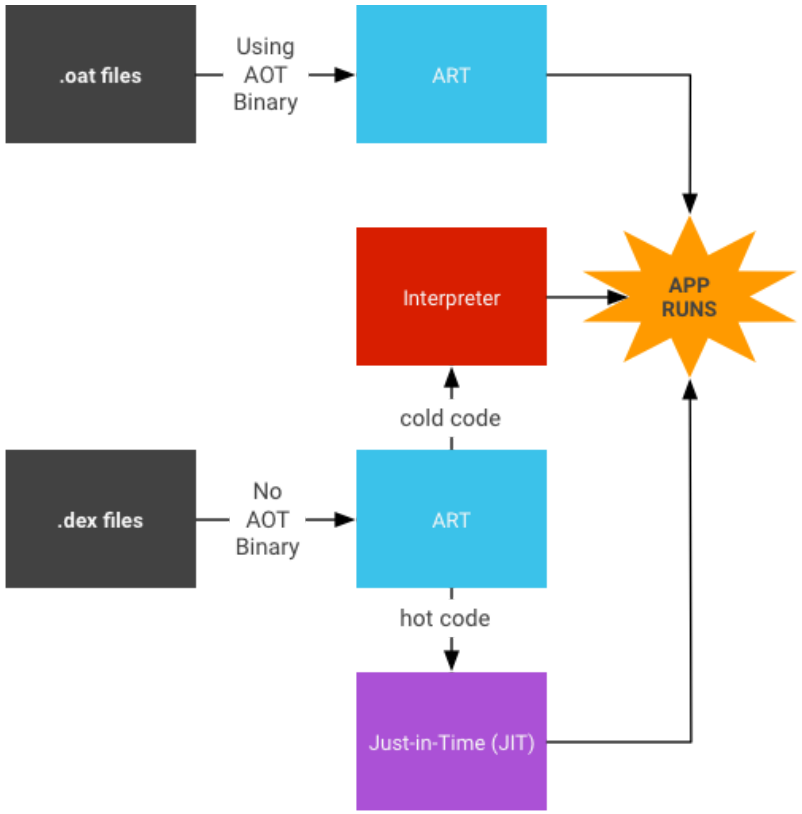
Compilation spécifique à l'architecture : Vous devez compiler à l'avance pour chaque architecture cible, ce qui peut augmenter la complexité de la distribution.

Comprendre l'architecture d'AOSP

ART et sa stratégie hybride:

- ART utilise une approche hybride combinant les avantages du JIT et de l'AOT. Lorsqu'une application est installée, elle est AOT compilée, ce qui signifie qu'elle est transformée en code machine natif pour une exécution rapide. Cependant, ART utilise également le JIT pour certaines parties du code qui ne sont pas fréquemment exécutées ou pour lesquelles une compilation AOT complète serait contre-productive en termes de temps ou d'espace.

- Cette stratégie hybride permet à ART d'offrir une performance optimale tout en conservant l'efficacité de la mémoire et de la batterie.



Comprendre l'architecture d'AOSP

- **Framework Android:**
- Le Framework Android est fondamentalement le cœur des fonctionnalités et services que les développeurs utilisent pour construire des applications Android. C'est une couche intermédiaire entre le système d'exploitation et les applications qui s'exécutent dessus.
  - **Composants principaux:**
    1. **Activities:** Gèrent l'interface utilisateur d'une application. Chaque activité correspond généralement à un écran dans l'application.
    2. **Services:** Exécutent des opérations en arrière-plan sans une interface utilisateur. Par exemple, un service pourrait jouer de la musique pendant que l'utilisateur est dans une autre application.
    3. **Content Providers:** Gèrent l'accès aux données. Ils offrent une interface abstraite pour partager des données entre différentes applications.
    4. **Broadcast Receivers:** Répondent aux annonces globales du système. Par exemple, une application peut être informée lorsque le système a terminé le démarrage ou lorsque le téléphone est sur le point de s'éteindre.
    5. **Intents:** Un mécanisme de messagerie qui permet de demander une action d'un autre composant, qu'il s'agisse d'une activité, d'un service, d'un fournisseur de contenu ou d'un récepteur de diffusion.

Comprendre l'architecture d'AOSP

- **Framework Android:**
  - **Fonctionnalités essentielles:**
    1. **Gestion des Fenêtres:** Le Framework Android possède un gestionnaire de fenêtres qui détermine comment les fenêtres (y compris les activités) sont affichées à l'écran. Cela concerne également la superposition des fenêtres, la gestion de dialogues, etc.
    2. **Notifications:** Les applications peuvent afficher des notifications pour informer l'utilisateur d'un événement spécifique, même si l'application n'est pas au premier plan. Le framework offre une API complète pour personnaliser ces notifications.
    3. **Ressources et Thèmes:** Gestion des ressources telles que les images, les chaînes de caractères et les mises en page. Les thèmes permettent une personnalisation de l'aspect et du comportement de l'UI.
    4. **Accès aux Capteurs:** Le Framework Android offre des API pour accéder à des capteurs tels que l'accéléromètre, le gyroscope, le capteur de proximité, etc.
    5. **Connectivité:** Gestion de la connectivité réseau, que ce soit pour le Wi-Fi, la 4G/5G, ou même les opérations Bluetooth.

Comprendre l'architecture d'AOSP - HAL

- **HAL:** La couche HAL (Hardware Abstraction Layer) d'Android sert de pont entre le code spécifique au matériel (souvent écrit en C ou C++) et le framework Android de haut niveau (écrit principalement en Java). La manière dont HAL est implémentée et interagit avec les composants logiciels et matériels est essentielle pour assurer la modularité et l'extensibilité d'Android.

1. **Définition d'interface** :
   Chaque type de périphérique matériel (par exemple, caméra, audio, capteur, etc.) a une interface définie. Dans les versions précédentes d'Android, ces interfaces étaient définies en tant que structures C/C++ dans des en-têtes. Avec l'introduction de HIDL (HAL Interface Description Language) et ensuite AIDL, ces interfaces sont définies dans des fichiers .hal ou .aidl.
2. **Implémentation du HAL** :
   Les fournisseurs de matériel fournissent des bibliothèques qui implémentent ces interfaces. Ces bibliothèques sont spécifiques à un matériel donné.
   Les bibliothèques HAL sont généralement des bibliothèques partagées (*.so) écrites en C ou C++.
3. **Chargement dynamique** :
   Lorsque le système a besoin d'accéder à un périphérique, il charge dynamiquement la bibliothèque HAL appropriée à l'aide de dlopen(), puis utilise dlsym() pour obtenir les pointeurs vers les fonctions.

Comprendre l'architecture d'AOSP - HAL

4. **Binder IPC** :
   Pour les services qui nécessitent une communication interprocessus (IPC), Android utilise le mécanisme Binder. Avec HIDL et AIDL, l'IPC Binder est fortement intégré pour permettre la communication entre le framework Java et les services HAL.

5. **Versioning** :
   HAL est conçu avec le versioning à l'esprit. Cela signifie que même si l'interface HAL pour un type de périphérique change dans une nouvelle version d'Android, elle reste compatible avec les anciennes implémentations.

6. **Treble et HIDL** :
   Avec Android 8.0 (Oreo), Google a introduit Project Treble, qui a modifié la manière dont HAL interagit avec le reste du système. Dans Treble, HAL est divisé en plusieurs processus distincts, chacun s'exécutant dans son propre bac à sable, pour des raisons de sécurité et de modularité.
   Avec Treble, l'introduction de HIDL permet de définir les interfaces HAL en utilisant HIDL, qui est compilé pour produire du code C++ et Java, permettant à la fois une communication interne et une communication avec le framework Android.

les outils AOSP

- **Outils de compilation**:

Lorsqu'il s'agit de construire AOSP, plusieurs outils et systèmes de construction sont essentiels pour le processus. Les deux principaux systèmes de construction que vous devez connaître sont Make et Soong

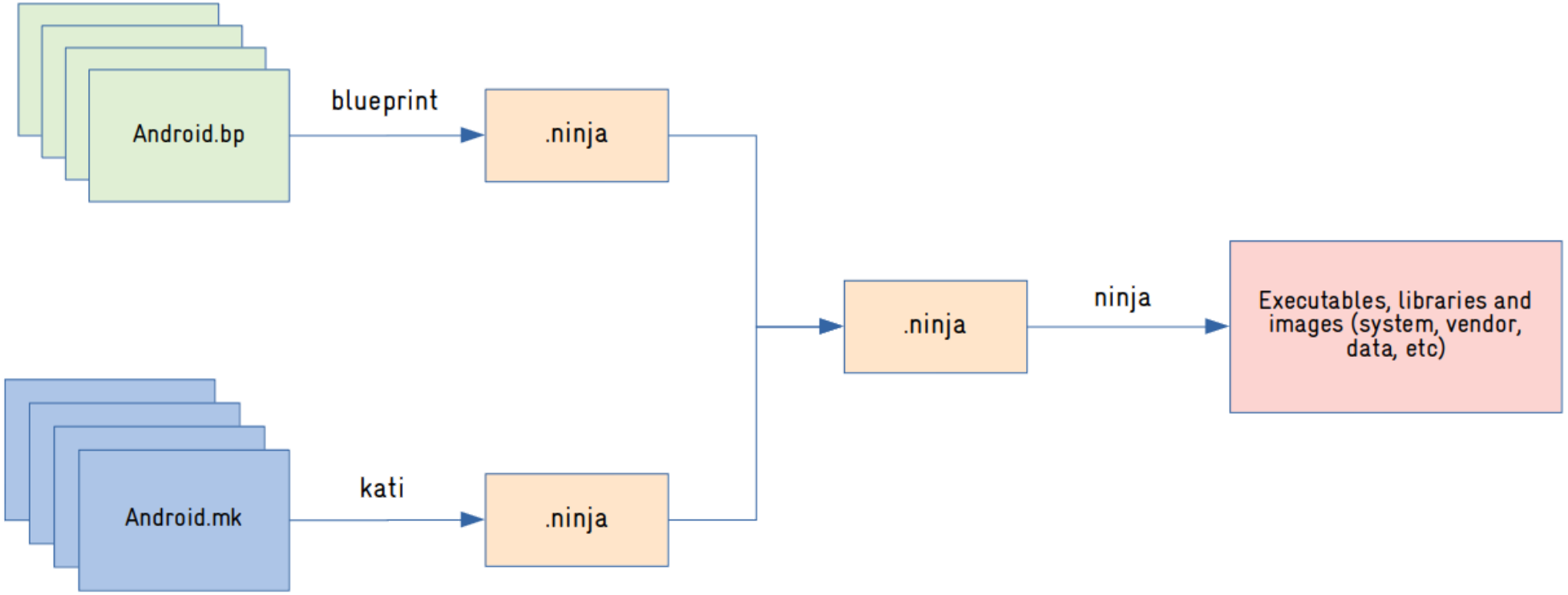
1. **Make:**
    - Dans les versions antérieures d'Android, le système de construction principal utilisé était basé sur GNU Make. Les instructions de construction étaient définies dans des fichiers nommés Android.mk.
    - Bien que puissant, avec le temps et la croissance du codebase Android, ce système est devenu moins adapté à la tâche. La nécessité de définir des dépendances correctes et de gérer une grande quantité de code source devenait complexe avec Make.

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)

LOCAL_MODULE := my_module
LOCAL_SRC_FILES := my_source_file.c

include $(BUILD_SHARED_LIBRARY)
```

Comprendre l'architecture d'AOSP



Comprendre l'architecture d'AOSP

- **Outils de compilation**:

Soong:

- Reconnaissant les limites de Make pour leurs besoins, Google a introduit Soong comme le nouveau système de construction pour Android.
- Soong est écrit en Go et est conçu pour être plus rapide, plus fiable et plus facile à maintenir que le système basé sur Make.
- Dans Soong, les fichiers de description de construction utilisent l'extension .bp (Blueprint).

```
cc_binary {
    name: "my_module",
    srcs: ["my_source_file.c"],
    shared_libs: ["some_shared_library"],
}
```

Exercice Soong

Objectif : Familiarisez-vous avec la structure de base de Soong pour les applications Android développées en Kotlin

1. **Préparation de l'environnement :** Créez un répertoire pour cet exercice, par exemple soong_kotlin_demo.
2. **Création des Fichiers Sources :** Dans ce répertoire, créez une simple activité Kotlin nommée MainActivity qui affiche un message "Hello from Soong and Kotlin!" à l'écran.
3. **Définir le fichier Blueprint de Soong :** Créez un fichier Blueprint de Soong pour cette application. Assurez-vous de définir correctement les sources, le manifeste, la version du SDK et les bibliothèques nécessaires pour Kotlin.
4. **Créer le fichier Manifest :** Rédigez un AndroidManifest.xml adapté pour votre activité.
5. **Instruction pour la construction :** Écrivez les commandes que vous utiliseriez pour construire cette application dans un environnement AOSP complet avec Soong.

Comprendre l'architecture d'AOSP

Exercice de correspondance

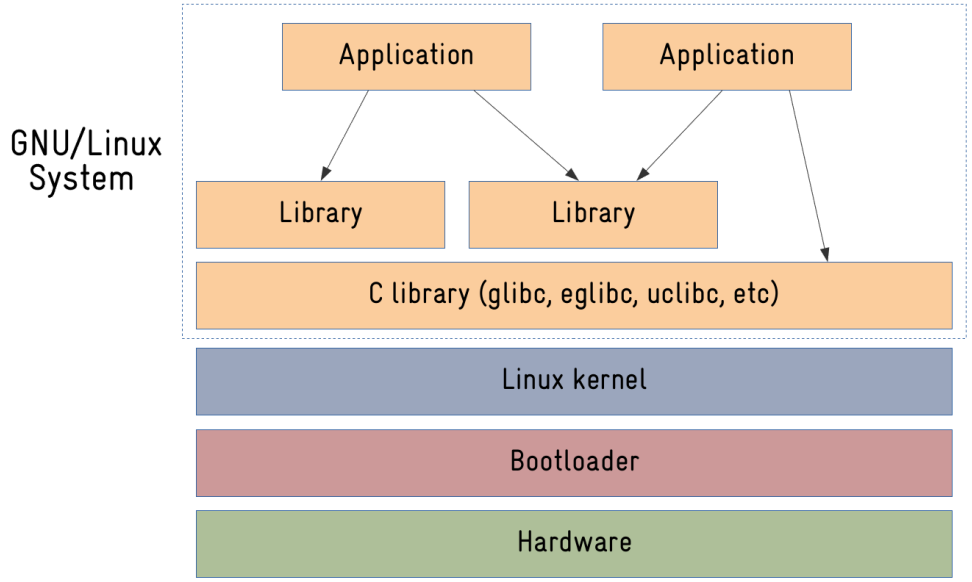
Liste d'éléments:

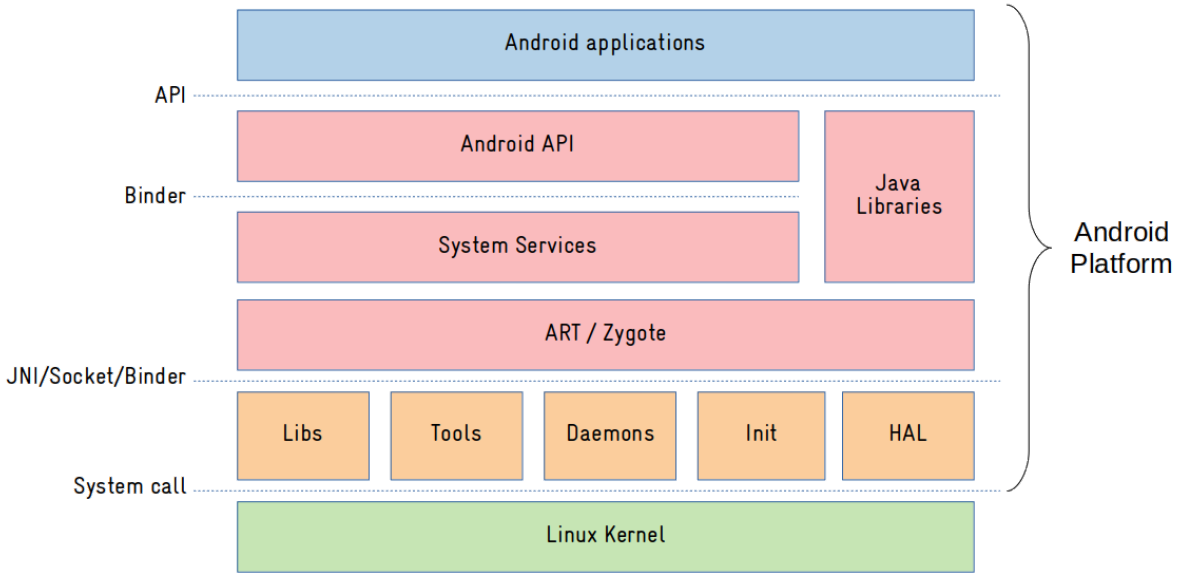
- Noyau Linux
- Outils de compilation
- Applications système
- Fichiers de structure
- HAL (Hardware Abstraction Layer)
- Binder IPC
- Frameworks
- ART (Android Runtime)

Liste de descriptions:

- Ils transforment le code source en exécutable pour la plateforme Android.
- Ces programmes sont intégrés dans l'OS et fournissent des fonctionnalités de base.
- Il s'agit du cœur du système d'exploitation et gère les ressources matérielles.
- Organisation et disposition des composants d'AOSP.
- Permet à Android de fonctionner indépendamment des spécificités du matériel.
- Mécanisme de communication inter-processus utilisé dans Android.
- Fournit les fonctions principales d'Android pour le développement d'applications.
- Environnement d'exécution qui remplace Dalvik, utilisé pour exécuter des applications Android.

Comprendre le noyau Linux utilisé par AOSP





Comprendre le noyau Linux utilisé par AOSP

- Le noyau Linux est le cœur du système d'exploitation Android, comme c'est le cas pour de nombreuses autres distributions Linux. C'est une couche logicielle fondamentale qui agit comme une interface entre le matériel de l'appareil et le logiciel s'exécutant dessus. Android, étant basé sur le noyau Linux, utilise ce noyau avec certaines modifications et ajouts spécifiques.

Les fonctionnalités du noyau Linux utilisé par AOSP

1. **Gestion des Processus et des Thread** : Le noyau gère le cycle de vie des processus et des threads, y compris leur création, planification et destruction
    - Le noyau Linux est conçu pour être préemptif, ce qui signifie qu'il peut interrompre un processus en cours d'exécution pour donner la priorité à un autre processus. C'est essentiel pour assurer une expérience utilisateur fluide sur un appareil Android, car de nombreuses applications peuvent s'exécuter simultanément
2. **Mémoire:** Assure la gestion de la mémoire physique, y compris la pagination, la segmentation et l'allocation de mémoire pour les processus
    - Le noyau Linux est conçu pour être préemptif, ce qui signifie qu'il peut interrompre un processus en cours d'exécution pour donner la priorité à un autre processus. C'est essentiel pour assurer une expérience utilisateur fluide sur un appareil Android, car de nombreuses applications peuvent s'exécuter simultanément

Comprendre le noyau Linux utilisé par AOSP

3. **Pilotes et Gestion des Périphériques**: Intègre des pilotes pour divers périphériques tels que les écrans tactiles, les caméras, le Wi-Fi, etc
    - Les pilotes sont des programmes qui permettent au noyau de communiquer avec le matériel. Dans Android, certains pilotes, comme ceux pour le GPU, peuvent être spécifiques, car ils doivent supporter des opérations graphiques intensives pour les jeux et les interfaces utilisateur
4. **Sécurité**: Fournit des mécanismes de sécurité tels que SELinux pour une granularité d'accès et de contrôle, et les espaces de noms pour isoler les processus
    - En plus des fonctionnalités de sécurité standard de Linux, Android a intégré SELinux en mode obligatoire, imposant des politiques de sécurité plus strictes pour les applications et les processus
5. **Système de Fichiers**: Supporte plusieurs systèmes de fichiers, dont ext4, f2fs, et d'autres adaptés aux dispositifs de stockage flash
    - Android a historiquement utilisé YAFFS2, mais est passé à ext4, qui supporte de plus grands volumes de stockage et offre de meilleures performances. Android utilise aussi F2FS pour certains appareils avec des mémoires flash

Comprendre le noyau Linux utilisé par AOSP

6. **Inter-process Communication (IPC)**: Mécanismes pour la communication entre processus, tels que les sockets, les sémasphores, et les files de messages
    - Bien que Linux offre des mécanismes IPC, Android a introduit Binder, qui est plus adapté aux besoins spécifiques des appareils mobiles.
7. **Wake Locks** : Une fonctionnalité spécifique à Android qui permet de garder le CPU allumé (ou d'autres ressources) même si l'appareil est en veille, pour permettre des opérations en arrière-plan.
8. **Binder** : Un mécanisme IPC unique à Android qui permet la communication entre processus et composants du système.
9. **Low Memory Killer** : Une adaptation pour Android qui tue les processus selon une certaine priorité lorsqu'il y a une faible disponibilité de mémoire.

Comment AOSP interagit avec le noyau Linux

- **Démarrage du Système (Boot Sequence)** :Lorsque l'appareil est allumé, le noyau Linux est chargé en premier. Une fois que le noyau a terminé son initialisation, il démarre le processus init d'Android qui démarre le reste du système

1. Amorçage du chargeur de démarrage (Bootloader): Lorsque l'appareil est allumé, le chargeur de démarrage est le premier code à s'exécuter. Il initialise le matériel, comme le CPU et la mémoire, et charge le noyau Linux dans la mémoire. Sur de nombreux appareils Android, le chargeur de démarrage se décompose en plusieurs étapes, souvent avec un "bootloader primaire" (PBL) et un "bootloader secondaire" (SBL).
2. Démarrage du noyau: Une fois chargé par le chargeur de démarrage, le noyau Linux démarre et initialise le système. Il configure les sous-systèmes essentiels tels que les contrôleurs de mémoire, les pilotes essentiels et d'autres interfaces. Le noyau décompresse également l'initramfs (système de fichiers initial en RAM) si nécessaire.
3. Exécution du processus init: Détails: init est le premier processus lancé par le noyau Linux. Dans Android, init est un binaire spécifique qui joue plusieurs rôles. Il monte les partitions nécessaires, lance des démons essentiels, configure les interfaces réseau, et établit des environnements pour d'autres processus. Il lit le fichier init.rc (et d'autres fichiers .rc associés) pour déterminer la séquence de démarrage et les services à lancer.

Comment AOSP interagit avec le noyau Linux

- **Démarrage du Système (Boot Sequence)**

4. Démarrage du Zygote: Une fois que le processus init a terminé l'initialisation initiale, il démarre le Zygote, qui est le processus parent de toutes les applications Android. Le Zygote précharge les classes Java communes pour accélérer le lancement des applications et attend les demandes de démarrage d'applications.
5. Lancement du système de serveur: Le système de serveur est le premier processus Java à s'exécuter sur un appareil Android. Il initialise le framework Android en démarrant des services essentiels tels que ActivityManagerService, PackageManagerService, et d'autres. Ces services sont cruciaux pour la gestion des applications, la manipulation des activités, et d'autres opérations au niveau du système.
6. Démarrage de l'interface utilisateur: Une fois que le système de serveur a démarré les services essentiels, l'interface utilisateur est lancée. Cela commence généralement par la démonstration de l'écran de verrouillage, suivi du lanceur d'applications. L'utilisateur peut alors interagir avec l'appareil.

Comment AOSP interagit avec le noyau Linux

2. **Interaction à travers la HAL (Hardware Abstraction Layer)** : AOSP utilise la HAL pour interagir avec le matériel. La HAL fournit des interfaces bien définies, et le noyau communique avec le matériel en utilisant les pilotes appropriés.
3. **Utilisation de Binder**: Le Framework Android communique avec des services système (tournant souvent dans d'autres processus) via le mécanisme Binder.
4. **Gestion des Ressources**: Android fait des appels au noyau pour des opérations telles que la création de threads, l'allocation de mémoire, et l'accès au réseau.
5. **Événements du Système**: Les événements matériels, comme un appui sur un bouton ou un signal d'une antenne Wi-Fi, sont envoyés au noyau. Celui-ci les transmet ensuite à l'espace utilisateur d'Android pour traitement.
6. **Mise à jour OTA (Over-The-Air)**: Lorsqu'une mise à jour du système est appliquée, elle peut également contenir des mises à jour pour le noyau. C'est le bootloader qui se charge de flasher le nouveau noyau sur l'appareil.

Comment AOSP interagit avec le noyau Linux Exercice

Objectif: Comprendre l'interaction entre le framework Android et le noyau Linux.

1. Binder & IPC:
    - Créez une application simple avec deux composants : une activité principale et un service Android.
    - Définissez une interface AIDL pour le service qui expose une méthode simple, par exemple, getBinderMessage(), qui retourne une chaîne de caractères.
      Depuis l'activité principale, connectez-vous au service et invoquez la méthode pour récupérer le message.
      Affichez le message à l'utilisateur.
2. Gestion des Ressources:
    - Dans la même application, créez un nouveau thread depuis l'activité principale.
    - À l'intérieur du thread, ouvrez une socket vers un serveur de votre choix (par exemple, httpbin.org) et récupérez une réponse.
    - Affichez la réponse dans l'activité principale.

Comment AOSP interagit avec le noyau Linux Exercice

3. Événements du Système:
    - Dans l'activité principale, ajoutez un gestionnaire d'événements pour intercepter l'appui sur les boutons de volume.
    - Lorsque l'utilisateur appuie sur un bouton de volume, affichez un message toast indiquant si le volume a été augmenté ou diminué.

AOSP Partie 2

Comprendre les services Android

- Les services Android jouent un rôle crucial dans le fonctionnement du système d'exploitation. Ils forment la couche de base qui permet au système et aux applications de fonctionner en parfaite harmonie, offrant ainsi une expérience utilisateur fluide.

Les différents services Android

1. **Gestionnaire de packages (PackageManagerService)**
    - Le PackageManagerService est responsable de la gestion de toutes les applications installées sur l'appareil. Cela inclut la maintenance des métadonnées des applications, des permissions, des signatures et plus encore.
    - **Démonstration:**
        - **Lister les applications installées** : Utilisez la commande `ADB adb shell pm list packages` pour afficher la liste des packages installés.
        - **Obtenir des détails sur une application** : `adb shell dumpsys package <nom du package>` pour afficher des informations détaillées sur une application particulière.

Comprendre les services Android

2. **Gestionnaire de fenêtres (WindowManagerService)** :
    - Le WindowManagerService gère tout ce qui est relatif à l'affichage des fenêtres sur l'appareil. Cela inclut les animations, les superpositions, la taille et la position des fenêtres.
    - **Démonstration :**
        - **Inspecter les propriétés des fenêtres** : Utilisez la commande ADB `adb shell dumpsys window` pour afficher les informations sur l'état actuel des fenêtres.
3. **Gestionnaire d'activités (ActivityManagerService)** :
    - Le ActivityManagerService est chargé de la gestion du cycle de vie des activités, de la gestion de la pile d'activités, et de la supervision des processus en arrière-plan.
    - **Démonstration :**
        - Lister les activités en cours d'exécution : Utilisez la commande `adb shell dumpsys activity activities` pour afficher la pile d'activités en cours.
        - Afficher la consommation mémoire des processus : `adb shell dumpsys meminfo`.

Comprendre les services Android

4. **Gestionnaire de contenu (ContentManagerService)** :
    - Ce service est responsable de l'accès aux données entre différentes applications à travers les fournisseurs de contenu.
    - **Démonstration** :
        - Accéder aux contacts : Montrez comment une application peut utiliser le fournisseur de contenu pour récupérer une liste de contacts à l'aide de l'API ContentResolver.
5. **Gestionnaire de notifications (NotificationManagerService)** :
    - Comme son nom l'indique, ce service est chargé de la gestion des notifications. Il permet aux applications d'afficher, de mettre à jour et de supprimer leurs notifications.
    - **Démonstration** :
        - Créer une simple notification : Montrez comment coder une application simple qui envoie une notification avec un texte, une icône et une action cliquable.

Comprendre les services Android

6. **Gestionnaire de puissance (PowerManagerService)** :
    - Gère la consommation d'énergie du dispositif, y compris la mise en veille, le réveil et l'acquisition de différents types de verrous de veille.
    - **Démonstration** :
        - Observer les wakelocks : Utilisez `adb shell dumpsys power` pour montrer comment certaines applications peuvent empêcher l'appareil de se mettre en veille.
7. **Gestionnaire de connectivité (ConnectivityManagerService)** :
    - Gère toutes les connexions réseau de l'appareil, y compris Wi-Fi, données mobiles, VPN, etc.
    - **Démonstration** :
        - Vérifier l'état de la connectivité : Montrez comment une application peut vérifier si elle est connectée à Internet ou non à l'aide du ConnectivityManager.

Comprendre les services Android

Interaction des services Android avec les autres éléments d'AOSP

1. **Avec le Noyau Linux** :
    - Les services Android, en particulier ceux liés au matériel comme le PowerManagerService et le ConnectivityManagerService, communiquent directement avec le noyau pour gérer les ressources matérielles.
    - Par exemple, la mise en veille et le réveil du dispositif nécessitent une communication étroite avec le noyau.
2. **Avec les applications**:
    - Les applications interagissent constamment avec les services Android pour réaliser leurs tâches. Par exemple, une application de messagerie pourrait utiliser le NotificationManagerService pour afficher une nouvelle notification.
    - Les applications peuvent également démarrer leurs propres services en arrière-plan, qui fonctionnent indépendamment des activités d'interface utilisateur.

Comprendre les services Android

3. **Avec le HAL (Hardware Abstraction Layer)** :
    - Certains services Android, en particulier ceux qui nécessitent un accès direct au matériel, communiquent avec le matériel via le HAL.
      Le HAL fournit des interfaces standardisées pour le matériel, garantissant que les services Android peuvent fonctionner sur une variété d'appareils avec des composants matériels différents.
4. **Avec le Framework Android** :
    - Les services Android sont étroitement intégrés au framework Android. En fait, ils peuvent être considérés comme une partie intégrante du framework.
      Ils fournissent des APIs backend pour de nombreuses fonctions du framework. Par exemple, le framework peut offrir une API pour envoyer une notification, mais c'est le NotificationManagerService qui exécute réellement cette tâche.

Comprendre les applications système

- Une application système, parfois appelée "app système" ou "app intégrée", est une application qui est pré-installée sur le système d'exploitation Android et qui ne peut généralement pas être désinstallée par l'utilisateur final sans privilèges root. Ces applications sont intégrées dans la ROM du système et sont généralement stockées dans le répertoire /system/app ou /system/priv-app.

Comprendre les applications système

- **Fonctionnalités essentielles** : Les applications système fournissent des fonctionnalités essentielles qui permettent au téléphone de fonctionner de manière optimale dès la première utilisation. Par exemple, le dialer (téléphonie), les messages (SMS) et les paramètres.
- **Intégration étroite avec le système**: En raison de leur nature intégrée, les applications système peuvent avoir des autorisations et des privilèges supplémentaires qui ne sont pas disponibles pour les applications tierces. Cela leur permet d'offrir des fonctionnalités qui ne pourraient pas être fournies par des applications téléchargées depuis le Play Store.
- **Sécurité**: Les applications système sont installées dans une partition en lecture seule, ce qui les rend plus résistantes à la modification malveillante. De plus, certaines applications dans /system/priv-app ont des autorisations spéciales qui ne sont pas accessibles aux applications utilisateur ordinaires.

Comprendre les applications système

- **Performance**: Étant donné que ces applications sont pré-installées, elles peuvent être optimisées spécifiquement pour le hardware sur lequel elles tournent, garantissant ainsi une meilleure performance.
- **Uniformité et Cohérence**: Les applications système garantissent une expérience utilisateur uniforme et cohérente sur tous les appareils qui exécutent le système d'exploitation Android, indépendamment de la marque ou du modèle.
- **Interaction avec les couches inférieures de l'AOSP**: Les applications système, telles que NetworkManagement et ConnectivityService, interagissent directement avec les couches HAL (Hardware Abstraction Layer) et Kernel pour gérer et contrôler le matériel du dispositif.

Comprendre les applications système

- Chaque application système d'AOSP a son propre répertoire de code source, situé dans le dépôt AOSP
- **SystemUI**: platform/frameworks/base/packages/SystemUI
- **Paramètres**: platform/packages/apps/Settings
  Téléphonie (Dialer): platform/packages/apps/Dialer
- **Messagerie**: platform/packages/apps/Messaging
  Launcher (par défaut, Launcher3): platform/packages/apps/Launcher3
- **Package Manager**: Le code principal de cette fonctionnalité se trouve dans platform/frameworks/base/services/core/java/com/android/server/pm
- ***Input Methods (LatinIME)***: platform/packages/inputmethods/LatinIME
- **ConnectivityService**: Ce n'est pas une application en soi, mais un service. Son code est dans platform/frameworks/base/services/core/java/com/android/server

Comprendre les applications système

Comment interagir avec ces applications

1. **Intent**: Les Intents sont le principal mécanisme de communication entre les applications et les composants d'applications dans Android. Vous pouvez démarrer une application système ou un service spécifique en envoyant un Intent approprié.
2. **Binder**: Android utilise Binder pour la communication interprocessus (IPC). De nombreuses applications système exposent des services qui peuvent être appelés par d'autres applications via IPC.
3. **Content Providers**: Certaines applications système, comme Contacts, exposent des données via des Content Providers qui peuvent être consultées à l'aide d'URI spécifiques.
4. **APIs publiques**: Android SDK fournit des API pour interagir avec presque toutes les fonctionnalités du système. Par exemple, pour envoyer un SMS, vous pouvez utiliser la classe SmsManager.
5. **Modification du code source**: Comme AOSP est open-source, vous pouvez directement modifier le code source de n'importe quelle application système pour ajuster son comportement.

Comprendre les applications système

Exercice

Ajoutez une nouvelle préférence dans l'application Paramètres pour activer ou désactiver une fonctionnalité fictive appelée "Mode SuperPower".

Comprendre la gestion de la mémoire

- La gestion de la mémoire est un aspect essentiel d'Android, le système d'exploitation open source sur lequel AOSP (Android Open Source Project) est basé. Comprendre cette gestion est crucial pour optimiser les performances et la fluidité des applications et du système lui-même

1. **Mémoire physique**: Il s'agit de la mémoire RAM (Random Access Memory) qui est disponible sur le dispositif. Android gère cette mémoire de manière dynamique pour s'assurer que les ressources sont utilisées efficacement.
2. **Mémoire virtuelle**: Comme la plupart des systèmes d'exploitation modernes, Android utilise un système de mémoire virtuelle pour séparer la mémoire physique (RAM) de la mémoire que les applications et le système d'exploitation pensent utiliser. La mémoire virtuelle permet à chaque application de fonctionner comme si elle disposait de sa propre mémoire privée.
3. **Gestion de la mémoire pour les applications**: Dalvik/ART: Dans les premières versions d'Android, la machine virtuelle Dalvik était responsable de l'exécution du bytecode des applications Android. Depuis Android 5.0 (Lollipop), Dalvik a été remplacé par ART (Android Runtime), qui précompile le bytecode en code machine natif lors de l'installation de l'application. ART est conçu pour mieux gérer la mémoire et offrir de meilleures performances.

Comprendre la gestion de la mémoire

4. **Garbage Collector**: Android utilise le ramasse-miettes (Garbage Collector) pour libérer la mémoire qui n'est plus utilisée par les applications. Ceci est essentiel pour prévenir les fuites de mémoire et garantir que les ressources sont disponibles pour les nouvelles applications.
5. **Low Memory Killer**: Lorsque le système est à court de mémoire, Android utilise le "Low Memory Killer" pour fermer les processus qui ne sont pas essentiels. Cela permet de libérer de la mémoire pour les applications actuellement en cours d'exécution.
6. **Priorités des processus**: Android attribue des priorités à chaque processus en fonction de son état actuel (en avant-plan, en arrière-plan, service, etc.). Cela permet au système de décider quels processus tuer en premier lorsqu'il est à court de mémoire.

Comprendre la gestion de la mémoire

7. **ZRAM (ou swap compressé)**: Android utilise ZRAM, un espace de swap compressé dans la RAM, pour augmenter virtuellement la quantité de mémoire disponible. Cela aide à améliorer les performances sur les appareils avec une mémoire limitée.
8. **Cache**: Android utilise un système de mise en cache pour stocker des données fréquemment utilisées. Cela accélère l'accès aux données et améliore les performances générales.
9. **Mémoire partagée (Ashmem)**: Android utilise Ashmem (Android Shared Memory) pour permettre à différentes applications de partager des données sans avoir à les copier plusieurs fois en mémoire.

Comprendre la gestion de la mémoire

Exercice

Objectif: Analyser l'utilisation de la mémoire d'un appareil Android.

1. Connectez un appareil Android à votre ordinateur avec le débogage USB activé.
2. À l'aide d'adb, récupérez les informations de `/proc/meminfo`. Quelle est la quantité totale de RAM sur l'appareil?
3. Utilisez encore adb pour obtenir les informations de `/proc/vmallocinfo`. Pouvez-vous identifier une entrée liée à `ioremap`?
4. En utilisant Android Studio, créez une application simple qui crée de nombreux objets en appuyant sur un bouton. Utilisez le Memory Profiler pour observer les effets sur la mémoire.
5. Expliquez la différence entre la mémoire physique et la mémoire virtuelle en vous basant sur vos observations.

Comprendre la sécurité

- La sécurité est l'un des piliers du système d'exploitation Android, et AOSP (Android Open Source Project) met en œuvre de nombreux mécanismes pour protéger les utilisateurs et les données

SELinux (Security Enhanced Linux)

- SELinux est un module de sécurité pour le noyau Linux qui offre un contrôle d'accès obligatoire. Android utilise SELinux pour renforcer le modèle de sécurité Android en confinant les processus à un domaine et en limitant les capacités à l'intérieur de ce domaine

Les principales entités de SELinux sont:

1. **Types** : chaque fichier, processus ou autre objet dans le système a un type SELinux associé.
2. **Domaines** : ce sont les types associés aux processus.
3. **Politiques** : ce sont des règles définissant les interactions entre différents types.

Modes SELinux:

1. **Enforcing** : Toutes les politiques sont appliquées et les violations sont bloquées et enregistrées.
2. **Permissive** : Les politiques ne sont pas appliquées, mais les violations sont enregistrées. C'est utile pour le débogage.

Comprendre la sécurité

Configurer une politique SELinux:

- Les politiques SELinux sont écrites dans un langage spécifique et compilées en un format binaire que le système utilise.
    - **Politiques sources** : Ces fichiers sont écrits dans le langage de politique SELinux et sont situés dans /system/sepolicy.
    - **Compiler la politique** : Android fournit un outil appelé checkpolicy pour compiler les politiques.
    - **Chargement d'une nouvelle politique** : Après avoir modifié et compilé une politique, elle doit être chargée. C'est généralement fait pendant le démarrage.

Comprendre le langage des politiques SELinux

1. **Types** : Au cœur de SELinux se trouvent les "types". Tout, des fichiers aux processus, a un type associé dans SELinux. Par exemple, le type httpd_t pourrait être associé à un serveur web.
2. **Domaines** : Les domaines sont des types associés aux processus. Lorsqu'un processus est en cours d'exécution dans un domaine particulier, cela détermine les ressources auxquelles il peut accéder.
3. **Étiquettes** : Il s'agit d'attributions de type (et d'autres attributs SELinux) à des objets, comme des fichiers ou des processus.

Comprendre la sécurité

Compilation et chargement de la politique

- Pour associer un processus à un domaine, nous utilisons ce qu'on appelle des "transitions de type". C'est essentiellement un mécanisme qui décrit comment un processus (lancé depuis un certain type/domaine) peut transiter vers un autre type/domaine lorsqu'il exécute un certain fichier.
- Une fois la politique rédigée, elle doit être compilée en un format binaire que SELinux peut utiliser. Vous utiliserez généralement l'outil checkpolicy pour cela. Ensuite, la politique compilée peut être chargée dans le système.

Conseils

- **Utilisez les outils d'audit** : Les outils d'audit SELinux peuvent vous aider à détecter les violations et à comprendre quelles permissions sont nécessaires.
- **Commencez petit** : Lorsque vous débutez, écrivez des politiques simples et étendez-les progressivement.
- **Testez en mode "permissive"** : Avant de mettre en place une nouvelle politique, testez-la en mode permissif pour voir quelles violations se produisent sans bloquer les opérations.

Comprendre la sécurité

Autorisations d'application

- Les autorisations Android déterminent quelles ressources système spécifiques une application peut accéder. Les autorisations sont déclarées par l'application et octroyées par l'utilisateur lors de l'installation ou lors de l'exécution de l'application.

Signature d'application

- La signature d'application garantit que chaque mise à jour d'application est issue de la même source et ne peut être modifiée par des tiers. Les applications doivent être signées pour être installées sur un appareil

Comprendre la sécurité

Exercice : Création d'une politique SELinux pour une nouvelle application

Contexte

Vous développez une nouvelle application pour Android nommée `CustomApp`. Cette application stocke ses fichiers de configuration dans `/data/customapp/config`. Vous devez écrire une politique SELinux pour garantir que seul `CustomApp` peut accéder à ce répertoire.

Instructions

1. Définissez un nouveau type pour l'exécutable `CustomApp`.
2. Définissez un nouveau domaine pour le processus `CustomApp` lorsqu'il est en cours d'exécution.
3. Écrivez une règle de transition pour assurer que lorsque `CustomApp` est lancé, il s'exécute dans le domaine que vous avez défini.
4. Écrivez les règles d'accès pour permettre à `CustomApp` de lire et d'écrire dans `/data/customapp/config`.
5. Testez votre politique en mode permissif.

AOSP Partie 3

Utilisation du HAL et HIDL

- HAL (Hardware Abstraction Layer) et HIDL (HAL Interface Definition Language) sont des composants du système d'exploitation Android qui facilitent la communication entre le code du système d'exploitation et les pilotes matériels.
- Avec l'introduction d'Android Oreo, Google a introduit HIDL pour améliorer la modularité et faciliter les mises à jour du système d'exploitation. HIDL permet de définir l'interface du HAL en utilisant un langage spécifique. Une fois l'interface définie avec HIDL, elle peut être utilisée pour générer du code pour le côté système et le côté matériel

```
package com.example.demo@1.0;

interface IDemo {

    enum Result : int32_t {
        SUCCESS = 0,
        ERROR_UNKNOWN,
        ERROR_INVALID_VALUE
    };

    setExempleParams(int32_t value) generates (Result result);
};
```

Utilisation du HAL et HIDL

- Une fois que l'interface avec HIDL est définie, le compilateur HIDL générera automatiquement du code à partir de cette définition pour plusieurs langages, notamment C++.

```
hidl-gen -o [OUT_DIR] -Lc++-impl -r com.example.demo:[HARDWARE_PATH] com.example.demo@1.0::IDemo
```

- Une fois les fichiers HIDL générés. Nous pouvons écrire notre implémentation en utilisant ces fichiers d'en-tête
- Enregistrement du service: Nous pourrons utiliser les fichiers init.rc appropriés pour s'assurer que le service est démarré à l'initialisation
- Génération du code java :

```
hidl-gen -o [OUT_DIR] -Ljava -r com.example.demo:[HARDWARE_PATH] com.example.demo@1.0::IDemo
```

Utilisation du HAL et HIDL

- L'intégration et l'utilisation des bibliothèques partagées (souvent .so pour "shared object" en Linux/Android) fournies par un fabricant dans le système AOSP (Android Open Source Project) se fait généralement à travers le mécanisme HAL (Hardware Abstraction Layer). Voici un aperçu simplifié du processus :

1. **Bibliothèques natives (*.so)** : Le fabricant fournit généralement des bibliothèques binaires pour une plate-forme spécifique. Ces bibliothèques sont précompilées et peuvent être insérées directement dans le système de fichiers approprié pour la plate-forme (par exemple, /system/lib/ ou /vendor/lib/).
2. **Définir le HAL** : Les définitions d'interfaces (comme les fichiers .hal que nous avons discutés précédemment) permettent de créer une abstraction pour le matériel. Cela permet au système Android d'interagir avec le matériel sans avoir à connaître les détails spécifiques de l'implémentation.
3. **Implémentation du HAL** : Pour chaque interface définie, il doit y avoir une implémentation correspondante qui utilise la bibliothèque fournie par le fabricant pour réaliser les fonctions spécifiées.
4. **Utiliser le Service Binder** : Les implémentations HAL s'exécutent généralement en tant que services binder. Le système Android peut alors faire des appels à ces services à travers le mécanisme binder.
5. **Appel depuis Java** : Une fois que le service binder est en place, les couches supérieures d'Android (par exemple, les services système ou les applications) peuvent interagir avec ces services via des proxy Java, générés à partir des définitions d'interfaces HAL.

Partition Android

- Chaque appareil Android divise son stockage interne en plusieurs partitions pour séparer, protéger et gérer efficacement les différentes sections du système d'exploitation et les données utilisateur.
- **Partition Boot** : Cette partition contient le noyau Android et un ramdisk, qui sont chargés en mémoire et exécutés au démarrage de l'appareil. Le noyau est le cœur du système d'exploitation, interagissant directement avec le matériel, tandis que le ramdisk contient des fichiers temporaires nécessaires au démarrage.
- **Partition Recovery** : C'est un mode de démarrage alternatif pour les appareils Android. Il contient un petit système d'exploitation et une interface utilisateur pour permettre la récupération et la mise à jour du système principal.
- **Partition System** : Elle contient le système d'exploitation Android, y compris tous les fichiers système et les applications préinstallées. Elle est montée en lecture seule pendant le fonctionnement normal pour protéger le système contre les modifications non autorisées.
- **Partition Vendor** : Introduite avec Android 8.0, elle contient des binaires et des pilotes spécifiques au matériel de l'appareil. Elle permet une séparation entre le code d'Android et le code spécifique au fabricant.

Partition Android

- **Partition Data** : C'est là que résident toutes les données utilisateur, les applications téléchargées, les paramètres, les messages, et d'autres données. Elle est chiffrée sur la plupart des appareils modernes pour la sécurité.
- **Partition Cache** : Utilisée pour stocker temporairement des fichiers système et des données d'application. Par exemple, lors des mises à jour du système, les fichiers téléchargés sont stockés ici avant d'être appliqués.
- **Partition Misc** : Stocke diverses informations système et paramètres, tels que les options de démarrage.
- **Partitions A/B** : Pour les appareils qui prennent en charge les mises à jour sans interruption, il existe deux ensembles de partitions (A et B). Lors d'une mise à jour, un ensemble est utilisé pour l'utilisation courante, tandis que l'autre est mis à jour en arrière-plan.

Exercices d'application

Exercice : Service de Gestion de Tasks (Tâches)
Objectif : Développez une application de service qui permet de gérer une liste de tâches. Le service permettra d'ajouter une tâche, de la supprimer, de la récupérer et de lister toutes les tâches existantes en utilisant AIDL.

1. Définir l'interface AIDL :

- Créez une interface AIDL, ITaskManager.aidl, avec les méthodes suivantes :
  - addTask(Task task) : pour ajouter une tâche.
  - removeTask(Task task) : pour supprimer une tâche.
  - Task getTask(int index) : pour récupérer une tâche spécifique.
  - List getAllTasks() : pour récupérer toutes les tâches.

De plus, définissez un objet Task avec deux champs : String name (nom de la tâche) et String description (description de la tâche).

2. Implémenter le Service :

- Créez un service TaskManagerService qui implémente les méthodes définies dans l'interface AIDL. Ce service gardera une ArrayList pour stocker les tâches.

Exercices d'application

3. Exposer le Service via Binder
4. Développer une Application Client
    - Créez une application client simple avec une interface utilisateur permettant d'ajouter, de supprimer, de visualiser et de lister les tâches. Cette application communiquera avec le service TaskManagerService via AIDL pour effectuer ces opérations.
5. Gérer les Permissions :
    - Assurez-vous que seules les applications ayant la permission appropriée puissent se lier et interagir avec votre service. Définissez une permission personnalisée dans le manifeste de votre service, par exemple, com.example.permission.ACCESS_TASK_SERVICE, et vérifiez cette permission dans votre service avant d'autoriser toute interaction.

Exercices d'application

Exercice : Service de Luminosité avec HIDL
Objectif : Développez un service HAL (Hardware Abstraction Layer) en utilisant HIDL (HAL Interface Definition Language) qui permet de gérer la luminosité d'un écran sur un appareil Android. Vous allez créer un HAL de luminosité et une implémentation basique pour simuler le contrôle de la luminosité. Ensuite, vous écrirez un service système pour interagir avec ce HAL.

1. Définir l'interface HIDL
2. Implémenter le HAL
3. Intégrer avec le framework Android
4. Écrire une application client