rationaliser la **programmation des microcontrôleurs** avec **ESP-IDF Privilege Separation**

Harshal Patil, Espressif

Cet article présente la séparation des privilèges dans les applications de microcontrôleurs avec l'ESP-IDF d'Espressif Systems. Cette méthode permet de répartir les micrologiciels dans un noyau protégé et dans une application utilisateur, ce qui simplifie le développement. Il décrit les étapes à suivre pour commencer, afin de rendre le développement des microcontrôleurs plus efficace.

En général, les applications sur microcontrôleurs (MCU) sont développées sous forme de micrologiciels monolithiques. Cependant, dans un système d'exploitation à usage général, il existe deux modes de fonctionnement : le mode noyau et le mode utilisateur. En mode noyau, le programme a un accès direct et illimité aux ressources du système, alors qu'en mode utilisateur, le programme d'application n'a pas d'accès direct aux ressources du système. Pour accéder aux ressources, un appel système doit être effectué.

L'idée principale de cette séparation est de faciliter le développement de l'application finale sans se préoccuper des changements sous-jacents dans le système, tout comme les applications de bureau ou mobiles : le système d'exploitation sous-jacent gère les fonctionnalités critiques, et l'application finale peut utiliser l'interface présentée par le système d'exploitation.

Séparation de privilèges de l'ESP

En général, toute application ESP-IDF sur un SoC Espressif est construite sous la forme d'un seul micrologiciel monolithique sans aucune séparation entre les composants de noyau centraux (système d'exploitation, réseau, etc.) et la logique « applicative » ou « commerciale ». Dans le cadre de la séparation des privilèges de l'ESP, nous divisons l'image du micrologiciel en deux fichiers binaires séparés et indépendants : l'application protégée et l'application utilisateur.

Pour commencer

Commençons à construire notre premier projet, Blink. La LED clignotante est l'équivalent de « Hello World » du monde des systèmes embarqués, démontrant la chose la plus simple que vous pouvez réaliser avec un microcontrôleur pour voir une sortie réelle. C'est parti !

Étape 0 : matériel requis pour le projet

- Une carte de développement ESP32-C3 ou ESP32-S3 avec une LED intégrée. Il est possible d'utiliser les kits de développement ESP32-C3-DevKitM-1 ou ESP32-S3-DevKitC-1. Nous choisirons l'ESP32-C3-DevKitM-1 pour notre projet.
- Un câble USB 2.0 (Standard-A vers Micro-B)
- Un ordinateur fonctionnant sous Windows, Linux ou macOS

Étape 1 : obtenir le projet ESP Privilege Separation

- Clonez le dépôt du projet ESP Privilege Separation.

```
git clone https://github.com/espressif/esp-privile-
ge-separation.git
```

Étape 2 : configuration de l'environnement ESP-IDF

- Clonez le dépôt du projet ESP-IDF.

```
git clone -b v4.4.3 --recursive https://github.com/
espressif/esp-idf.git
```

- Installer les outils.

```
cd esp-idf
./install.sh
```

- Définissez les variables d'environnement.

```
source ./export.sh
```

- Appliquez le correctif ESP Privilege Separation spécifique à ESP-IDF.

```
git apply -v /idf-patches/
privilege-separation_support_v4.4.3.patch
```

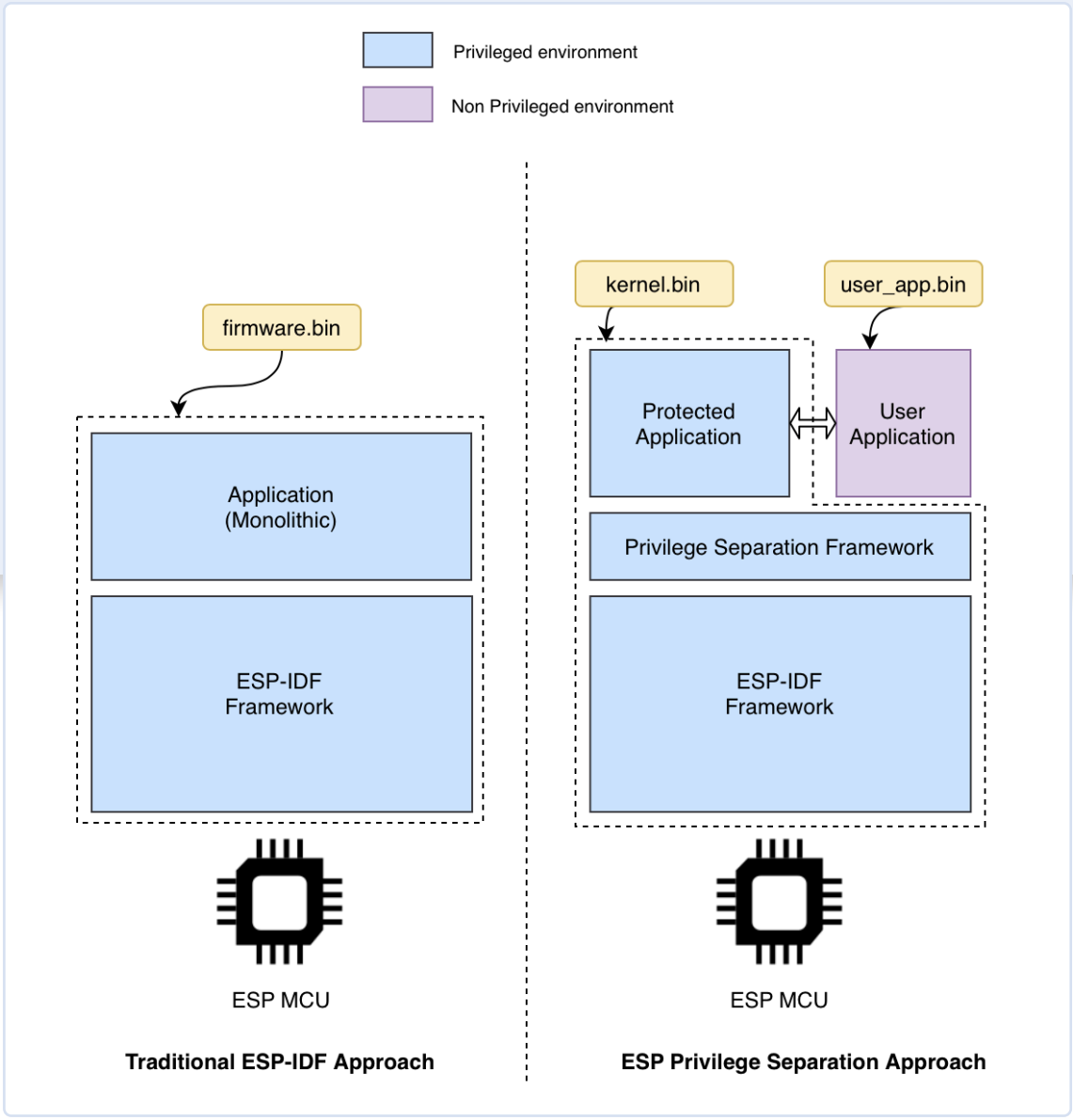


Figure 1. Comparaison d'un ESP-IDF traditionnel composé d'un micrologiciel monolithique unique par rapport au cadre ESP Privilege Separation, où l'image du micrologiciel est divisée. (Source : Espressif)

Étape 3 : exécution de l'exemple *Blink*

- Construisez l'exemple.

```
cd /examples/blink
idf.py set-target esp32c3
idf.py build
```

- Flashez et exécutez l'exemple.

```
idf.py flash monitor
```

Plonger dans l'implémentation

Notre LED clignote maintenant, mais pourquoi ce projet se distingue-t-il de tous les autres projets de clignotement ? Réponse : La séparation des privilèges. Essayons de comprendre l'implémentation :

- L'exemple est divisé en deux applications : l'application sécurisée et l'application utilisateur.
- L'application sécurisée recherche une partition utilisateur dans la table des partitions et charge l'application dans les sections définies par l'utilisateur.
- Elle configure ensuite les zones mémoire pour la région avec des privilèges inférieurs (`WORLD1`) et fournit l'accès tel que configuré par le développeur.
- Enfin, elle lance une tâche distincte à faible privilège avec le point d'accès de l'utilisateur.
- Une fois l'application utilisateur chargée, une tâche est lancée pour faire clignoter la LED connectée à GPIO8 (ESP32-C3-DevKitM-1).

Intégration avec ESP RainMaker

Nous avons maintenant notre devkit de clignotement de LED, mais nous ne sommes pas encore en mesure de contrôler le clignotement de la LED. Pour le qualifier d'appareil véritablement « connecté » et contrôler la LED, nous devons l'intégrer à ESP RainMaker. ESP RainMaker est un logiciel Cloud AIoT léger qui permet aux utilisateurs de construire, développer et déployer des solutions AIoT personnalisées avec un minimum de code et un maximum de sécurité. Commençons par construire l'exemple `rmaker_switch` présent dans le projet ESP Privilege Separation.

Étape 0 : exigences

- ESP-IDF pour ESP Privilege Separation a déjà été configuré avant la réalisation de tout exemple.
- Application mobile ESP RainMaker (Android/iOS).
- Réseau Wifi.

Étape 1 : installation d'ESP RainMaker

- Clonez le dépôt du projet ESP RainMaker.

```
git clone --recursive https://github.com/espressif/
esp-rainmaker
git checkout 00bcf4c0c30d96b8954660fb396ba313fb6c886f
export RMAKER_PATH=
```

Étape 2 : exécution de l'exemple rmaker_switch

- Construisez l'exemple.

```
cd /examples/rmaker_switch
idf.py set-target esp32c3
idf.py build
```

- Flashez et exécutez l'exemple.

```
idf.py flash monitor
```

Étape 3 : approvisionnement de l'appareil

- Une fois l'exemple exécuté, un code QR apparaît dans le terminal. Le code QR peut être utilisé pour l'approvisionnement sur Wifi.
- Connectez-vous à l'application mobile ESP RainMaker et cliquez sur *Add Device*. Cela ouvrira l'appareil photo pour scanner le code QR.
- Suivez le processus d'approvisionnement pour que votre appareil puisse se connecter à votre réseau Wifi.

Étape 4 : contrôle du commutateur de LED

- Enfin, vous verrez un appareil *Switch* ajouté à l'écran d'accueil de l'application mobile.
- Vous pouvez maintenant essayer de basculer l'icône du commutateur pour contrôler votre LED.

Mais comment avons-nous réalisé cette intégration ESP RainMaker transparente ? Nous introduisons un composant `rmaker_syscall` dans notre exemple Blink, qui révèle les appels système pour le framework ESP Rainmaker, et, lorsque l'application utilisateur démarre, il initialise le service ESP Rainmaker dans l'application protégée, et crée un appareil de commutation ESP RainMaker. Comme le composant ESP RainMaker est placé dans l'application protégée, des appels système sont nécessaires pour toutes les API publiques qu'il expose. Les fonctions d'extensibilité simples d'ESP Privilege Separation permettent d'ajouter des appels système personnalisés spécifiques à l'application, et cette couche utilise les données stockées dans le contexte de l'appareil pour exécuter les rappels de lecture et d'écriture dans l'espace utilisateur.

Mises à jour OTA du micrologiciel avec ESP Privilege Separation

La mise à jour du micrologiciel à distance (OTA) est l'une des fonctions les plus importantes pour tout appareil connecté. Elle permet aux développeurs de fournir de nouvelles fonctions et des corrections de bogues en mettant à jour l'application à distance. Dans ESP Privilege Separation, il y a deux applications – *protected_app* et *user_app*, pour lesquelles le framework permet des mises à jour indépendantes des deux fichiers binaires. L'application protégée, ayant des privilèges plus élevés, peut être mise à jour en suivant simplement l'interface OTA normale fournie par ESP-IDF, tandis que la mise à jour OTA de l'application utilisateur ayant des privilèges moins élevés est rendue possible car l'ESP Privilege Separation expose un appel système pour l'application utilisateur, `usr_esp_ota_user_app()`, pour exécuter le processus OTA.
Essayons l'exemple de mise à jour OTA de l'application utilisateur du projet ESP Priviledge Separation.

Étape 0 : exigences

- ESP-IDF pour ESP Privilege Separation a déjà été configuré avant la construction de tout exemple.
- Une URL publique de l'image de la nouvelle application utilisateur hébergée.

Étape 1 : exécution de l'exemple `rmaker_switch`

- Construisez l'exemple.

```
cd /examples/esp_user_ota
idf.py set-target esp32c3
idf.py build
```

- Flashez et exécutez l'exemple.

```
idf.py flash monitor
```

Étape 2 : démarrer la mise à jour OTA

Saisissez l'URL de l'OTA lorsque l'application utilisateur accepte l'OTA en utilisant la commande `user-ota` via la console. Cela déclenche une OTA et la nouvelle application utilisateur démarre une fois l'OTA terminée. Nous avons maintenant un appareil connecté avec des fonctions clés prêtes à être déployées.
Vous pouvez scanner le code QR pour regarder l'exemple ESP Privilege Separation, démontrant un cas d'utilisation réel de mise à jour OTA d'une application utilisateur à l'aide d'ESP Rainmaker et d'ESP Privilege Separation.

Comme nous l'avons dit, le succès d'un produit est déterminé par ses utilisateurs, et c'est vous ! Nous aimons savoir ce que vous pensez, alors n'hésitez pas à nous faire part de vos expériences, de vos suggestions et de vos commentaires. Vos commentaires alimentent notre innovation et nous aident à affiner notre produit pour mieux répondre à vos besoins. Visitez notre site web [1], envoyez-nous un courriel, connectez-vous avec nous sur les réseaux sociaux pour partager vos idées ou ouvrez un nouveau problème dans le dépôt GitHub du projet [2]. Si vous avez un besoin spécifique qui, selon vous, s'inscrit bien dans ce cadre, ou si la résolution de tels problèmes vous passionne, nous serions heureux de discuter avec vous d'une éventuelle collaboration.

230598-04

Questions ou commentaires ?

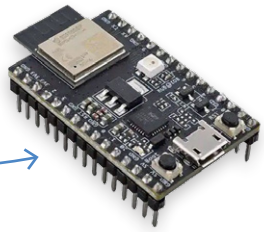
Envoyez un courriel à l'auteur (harshal.patil@espressif.com) ou contactez Elektor (redaction@elektor.fr).

À propos de l'auteur

Harshal Patil est ingénieur logiciel chez Espressif Systems depuis un an. Il est passionné par la résolution de problèmes concrets en construisant des systèmes sécurisés et connectés. Passionné de technologie et de sport, il aime explorer de nouveaux gadgets innovants et jouer au football dans son temps libre.

Produits

- **ESP32-C3-DevKitM-1**
  www.elektor.fr/20324
- **ESP32-S3-DevKitC-1**
  www.elektor.fr/20697



LIENS

[1] Espressif : https://espressif.com
[2] Le dépôt du projet sur GitHub : https://github.com/espressif/esp-privilege-separation/issues