PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE RORAIMA
DISCIPLINA: SISTEMAS OPERACIONAIS – DCC403
PROFESSOR: HEBERT ROCHA

DESENVOLVIMENTO DE DRIVER ANDROID PARA ATIVAÇÃO DA LANTERNA COM TEMPORIZADOR

AUTORES:
Guilherme Ramos
Kauã Victor
Lucas Nobre

Agosto de 2025
Boa Vista/Roraima

Sumário

1 Introdução

O presente relatório detalha a tentativa de desenvolvimento de um driver para o sistema operacional Android, cujo objetivo era ativar a lanterna (flash LED) de um dispositivo de forma automática, 30 segundos após a inicialização do sistema. Conforme o escopo do projeto, a principal restrição era realizar esta tarefa sem a utilização de um aplicativo de usuário (APK), exigindo uma solução a nível de sistema.

A abordagem planejada envolvia a criação de um executável nativo em C, a interação com a Camada de Abstração de Hardware (HAL) da câmera e sua integração ao processo de boot do sistema. Embora o plano teórico fosse sólido e as etapas iniciais tenham sido executadas, a implementação prática encontrou barreiras técnicas significativas que impediram a conclusão bem-sucedida do projeto. Este documento, portanto, serve como um registro do processo, das dificuldades enfrentadas e de uma análise detalhada sobre o caminho que seria necessário para alcançar o objetivo final.

2 Metodologia Planejada

A metodologia definida para o projeto seguia uma abordagem incremental, partindo da configuração do ambiente até a integração final do driver. As etapas planejadas estão descritas abaixo.

2.1 Proposta de Implementação do Driver Nativo

O núcleo do projeto seria um programa nativo em C (`flash_hal.c`) projetado para interagir com o HAL da câmera. O código proposto realizaria as seguintes ações:

1. Pausar a execução por 30 segundos com a função `sleep(30)`.
2. Obter acesso ao módulo de hardware da câmera via `hw_get_module()`.
3. Ativar o modo tocha (lanterna) através da função `set_torch_mode()` exposta pela interface do HAL.

O código-fonte proposto para esta etapa está listado abaixo.

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <hardware/hardware.h>
4  #include <hardware/camera.h>
5
6  int main() {
7      sleep(30);
8
9      camera_module_t* camera_module = NULL;
10     if (hw_get_module(CAMERA_HARDWARE_MODULE_ID, (const hw_module_t**)&
       camera_module) == 0) {
11
12         camera_device_t* camera_device = NULL;
13         // Abertura do dispositivo de camera. O ID "0" refere-se a
       camera principal.
14         // O cast para (struct hw_device_t**) eh a forma correta de
       passar o ponteiro.
```

```
        if (camera_module->open(camera_module, "0", (struct hw_device_t
    **)&camera_device) == 0) {

            if (camera_device && camera_device->ops->set_torch_mode) {
                camera_device->ops->set_torch_mode(camera_device, 1); //
     1 = LIGADO
            }

            // Fechamento correto do dispositivo
            camera_module->close((hw_device_t*)camera_device);
        }
    }
    return 0;
}
```

Listing 1 – Proposta de código-fonte para o driver nativo (flash_hal.c)

2.2 Proposta de Integração com o Sistema de Inicialização

A automação do driver seria realizada através do sistema ‘init‘ do Android. Um novo serviço seria adicionado a um dos arquivos de inicialização (‘.rc‘), como ‘init.rc‘ ou ‘init.goldfish.rc‘, para executar o binário nativo durante o boot.

```
service flash_on_boot /system/bin/flash_hal
    class main
    user root
    group camera
    oneshot
```

Listing 2 – Proposta de definição do serviço no arquivo init.rc

3 Dificuldades e Barreiras para a Conclusão do Projeto

Durante a tentativa de implementação, foram encontradas dificuldades progressivas. As iniciais foram contornadas, mas as barreiras subsequentes se mostraram impeditivas para a conclusão do projeto com os recursos disponíveis.

3.1 Dificuldades Iniciais (Solucionadas)

3.1.0.1 Configuração do Ambiente (ADB e NDK):

A primeira barreira foi a configuração do ambiente de desenvolvimento. O comando ‘adb‘ não foi encontrado inicialmente, exigindo a instalação e configuração do Android SDK. Posteriormente, a compilação do código nativo com o NDK falhou devido a caminhos incorretos do compilador (‘clang‘), o que demandou uma investigação manual para localizar a toolchain correta e ajustar o comando de compilação para a arquitetura de destino.

3.1.0.2 Abstração de Hardware:

Tentativas de controle via ‘adb shell‘ foram infrutíferas, o que confirmou que o acesso ao hardware no Android é protegido e abstraído, reforçando a necessidade de interagir com o HAL.

3.2 Barreiras Impeditivas (Não Solucionadas)

Após superar os desafios iniciais, o projeto se deparou com um obstáculo fundamental que impediu sua conclusão.

3.2.0.1 Necessidade de Compilação Completa do AOSP:

A abordagem de simplesmente compilar um binário nativo e inseri-lo em uma imagem de emulador pré-compilada (`adb push`) se mostrou inviável. As imagens de sistema fornecidas pelo Android Studio são otimizadas e protegidas (mesmo com acesso root). Modificações no sistema de arquivos, especialmente em partições como `/system`, e a adição de novos serviços no `init.rc` podem ser revertidas ou ignoradas por mecanismos de segurança como o **dm-verity**.

A solução correta e robusta para integrar um driver nativo é obter o código-fonte completo do Android (AOSP), adicionar o nosso código-fonte (`flash_hal.c`) e a definição do serviço (`init.rc`) diretamente na árvore de fontes e, então, compilar uma imagem de sistema (`system.img`) customizada a partir do zero.

3.2.0.2 Recursos Computacionais Insuficientes:

A compilação do AOSP é uma tarefa que exige recursos computacionais massivos. Os requisitos mínimos recomendados são:

- **Espaço em Disco:** O download do código-fonte do AOSP consome mais de 400 GB, e o processo de compilação pode exigir um total de 500 GB a 1 TB de espaço livre.
- **Memória RAM:** Recomenda-se no mínimo 16 GB, sendo 64 GB o ideal para um processo de compilação mais rápido.
- **Poder de Processamento:** A compilação pode levar várias horas, mesmo em máquinas potentes.

A falta de espaço em disco e de outros recursos computacionais foi a barreira final e intransponível que impediu o prosseguimento e a conclusão do projeto.

4 Conclusão e Análise Final

Embora a implementação final do driver de lanterna não tenha sido concluída, o projeto proporcionou um aprendizado profundo e valioso sobre a arquitetura interna do Android e os desafios práticos do desenvolvimento de software de baixo nível.

Ficou claro que, para modificar o comportamento do sistema operacional de forma nativa e persistente, não basta ter acesso root a uma imagem pré-existente. O caminho correto envolve a obtenção, modificação e compilação do código-fonte completo do sistema (AOSP). As dificuldades enfrentadas, desde a configuração do NDK até a barreira imposta pelos requisitos de hardware para compilar o AOSP, ilustraram de forma concreta a complexidade envolvida na criação de sistemas embarcados.

Conclui-se que, apesar do resultado prático não ter sido alcançado, o objetivo pedagógico foi plenamente atingido. O grupo demonstrou compreensão teórica da solução,

produziu o código-fonte que seria utilizado e diagnosticou com precisão os motivos técnicos que impediram a sua integração final, o que constitui uma experiência de aprendizado fundamental na área de Sistemas Operacionais.

Referências

GOOGLE. *Android Open Source Project*. Disponível em: <https://source.android.com/?hl=pt-br>. Acesso em: 7 ago. 2025.

C4 ANDROID. *Build Custom Android OS | Build Android ROM | Setup AOSP and Build Custom Android Image*. YouTube, 2023. Disponível em: <https://www.youtube.com/watch?v=vX8t9l8gnT0&t=1286s>. Acesso em: 7 ago. 2025.

MUKESH OTWANI. *How to Compile and Run Native C Code in Android Emulator*. YouTube, 2022. Disponível em: <https://www.youtube.com/watch?v=-OePyL55rvs&t=461s>. Acesso em: 7 ago. 2025.

GEEKS TRICK. *Flashlight with Android HAL - Android Source Tree*. YouTube, 2022. Disponível em: <https://www.youtube.com/watch?v=lIXWGU7YxWY&t=27s>. Acesso em: 7 ago. 2025.

FOOTPRINT TECHNOLOGY. *How to Build AOSP from Scratch*. YouTube, 2022. Disponível em: <https://www.youtube.com/watch?v=hBEAyXze9Ek&t=212s>. Acesso em: 7 ago. 2025.

PUNEETH S. *Flashlight implementation in Android with C code*. YouTube, 2022. Disponível em: <https://www.youtube.com/watch?v=cr8qon-4B1c>. Acesso em: 7 ago. 2025.

ABHISHEK PANDEY. *Controlling Flashlight in Android through native driver*. YouTube, 2023. Disponível em: <https://www.youtube.com/watch?v=hthdMsyMpdk>. Acesso em: 7 ago. 2025.

ANDROID DEVS. *Flashlight C HAL integration for Android*. YouTube, 2023. Disponível em: <https://www.youtube.com/watch?v=FdLNDdgehGY>. Acesso em: 7 ago. 2025.