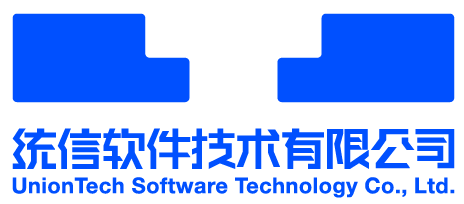
DKMS优化设计说明书



|  |  |  |  |
| --- | --- | --- | --- |
| 配置项编号 |  | | |
| 密级 | 秘密 | 版本 | V1.0 |
| 拟制 |  | 日期 |  |
| 审核 |  | 日期 |  |
| 批准 |  | 日期 |  |

**版本变更记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **修订说明** | **修订人** | **修订时间** |
| 1.0 | 创建 | 晋孝博 | 2022-07-28 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**目录**

[1. 概述 3](#_Toc541075897)

[1.1. 目的 3](#_Toc2013395244)

[1.2. 术语说明 3](#_Toc1989790142)

[2. 优化原理 4](#_Toc2085200067)

[2.1. 优化背景 4](#_Toc1574690931)

[2.2. 优化原理 4](#_Toc172583982)

[3. 打包方式 6](#_Toc194839509)

[3.1. 包中的内容 6](#_Toc928759459)

[3.2. ko\_dep\_kabi.txt 的说明 7](#_Toc1024436337)

# 概述

## 目的

本文档是针对DKMS优化的设计文档 ，在本文档中将给出原理，方案的设计。

本文档的适用读者为内核、系统、设计人员、开发人员、测试人员以及后续维护人员。

## 术语说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 缩语 | 英文全称 | 中文定义 |
| 1 | DKMS | Dynamic  Kernel Module Support | 动态内核模块支持 |
| 2 | KABI | Kernel Application Binary Interface | 内核二进制应用接口 |
|  |  |  |  |
|  |  |  |  |

# 优化原理

## 优化背景

动态内核模块支持 （Dynamic Kernel Module Support，DKMS）是用来生成Linux的内核模块的一个框架，其源代码一般不在Linux内核源代码树。 当新的内核安装时，DKMS支持的内核设备驱动程序 到时会自动重建。 DKMS可以用在两个方向：如果一个新的内核版本安装，自动编译所有的模块，或安装新的模块（驱动程序）在现有的系统版本上，而不需要任何的手动编译或预编译软件包需要。在dkms的两个应用方向上，都需要重新编译模块，模块的重新编译时间较长，导致用户等待时间较长。

内核导出的符号以及数据结构布局和版本信息构成了内核模块二进制接口，通常称为 kABI。模块通过访问内核或其他模块导出的符号与内核交互。 编译内核时，假定所有模块都与内核的静态部分一起编译。 这允许所有数据结构、内核配置、编译器版本和编译器标志一致，因此所有模块和静态内核可以一起工作。

## 优化原理

1. 为什么dkms会重新编译？

当第一次安装dkms包时， dkms包中是源码文件，需要根据内核的kabi，编译成可执行文件。

当内核升级时，dkms为了保证模块能够正常运行，会删除旧的可执行文件，重新根据新的内核kabi，编译成新的可执行文件。

1. 优化的依据

对于上游linux内核而言，它的更新速度极快，不能也不会保证kabi的兼容性。

但是对于UOS内核而言，我们的更新速度较慢，能够也有必要保持kabi的兼容性。

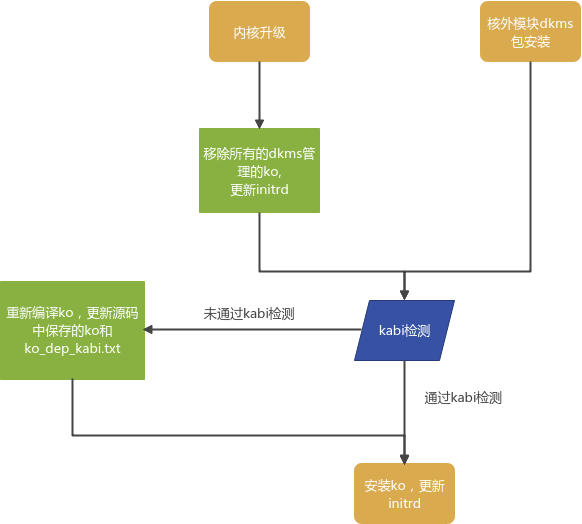
因此我们在dkms 中加入一套kabi检测机制，如果dkms管理的模块所依赖的内核kabi随着内核的升级并没有发生变化，则不需要重新编译，如果发生了变化，则重新编译。

1. 优化的方式

我们依赖核外模块的dkms包中的ko\_dep\_kabi.txt 文件和已经为特定内核版本编译好的ko以及内核头文件中的Module.symvers文件。

通过查询ko\_dep\_kabi.txt 中的kabi和Module.symvers中的kabi版本是否一致，来判断内核是否兼容之前的ko。 如果相同，不会重新编译，直接将编译好的ko，放到指定目录。如果不同，重新编译，重新编译时，会自动更新ko\_dep\_kabi.txt 和 dkms包中的ko。

1. kabi兼容性检测流程



# 打包方式

## 核外模块dkms包中的内容

1. uos 下的核外模块的dkms包，与普通的dkms下内容相比，为了减少安装和内核升级的时间，核外模块的dkms需要包含已经编好的ko和模块所依赖的内核kabi文件ko\_dep\_kabi.txt，这么做本质上是以空间换时间。

需要包含的ko如下，考虑这个核外模块的dkms主要会用在哪些内核版本和架构上。

创建对应内核版本和架构的文件夹，里面要包含编译好的ko和ko\_dep\_kabi.txt

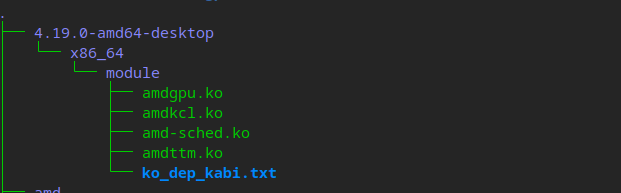
以amdgpu dkms 为例：

在源码下要包含这样一个目录，4.19.0-amd64-desktop/x86\_64/module/

该目录下包含，amdgpu.ko amdkcl.ko amd-sched.ko amdttm.ko ko\_dep\_kabi.txt

4.19.0-amd64-desktop 是 内核发行号（uname -r）, x86\_64 是架构，然后是module目录。

则该dkms包支持在4.19.0-amd64-desktop内核，x86\_64架构上的优化。



如果要同时支持其他内核版本和架构，就要按照上述的结构，再创建目录。

## ko\_dep\_kabi.txt 的说明

ko\_dep\_kabi.txt 是模块依赖的kabi和其modversion构成的文件

1. 如果该dkms包只会生成一个ko,则直接使用

modprobe --dump-modversions 生成ko\_dep\_kabi.txt即可。

例如 modprobe --dump-modversions ./test.ko > ko\_dep\_kabi.txt

1. 如果该dkms包会生成几个ko， ko\_dep\_kabi.txt 文件的生成就复杂些。

原因在于其中ko有依赖关系，一些ko依赖另一些ko生成的kabi，这些kabi和内核没关系，不能写到ko\_dep\_kabi.txt 中。

因此，在通过modprobe --dump-moversions 依次将这些ko依赖的kabi写入到ko\_dep\_kabi.txt 后，需要剔除这些ko自己生成kabi。可以使用nm命令，来查看模块自己生成的符号,也就是kabi，然后将其从ko\_dep\_kabi.txt 去除。

我们后期会提供一个自动化工具，来帮助生成ko\_dep\_kabi.txt。