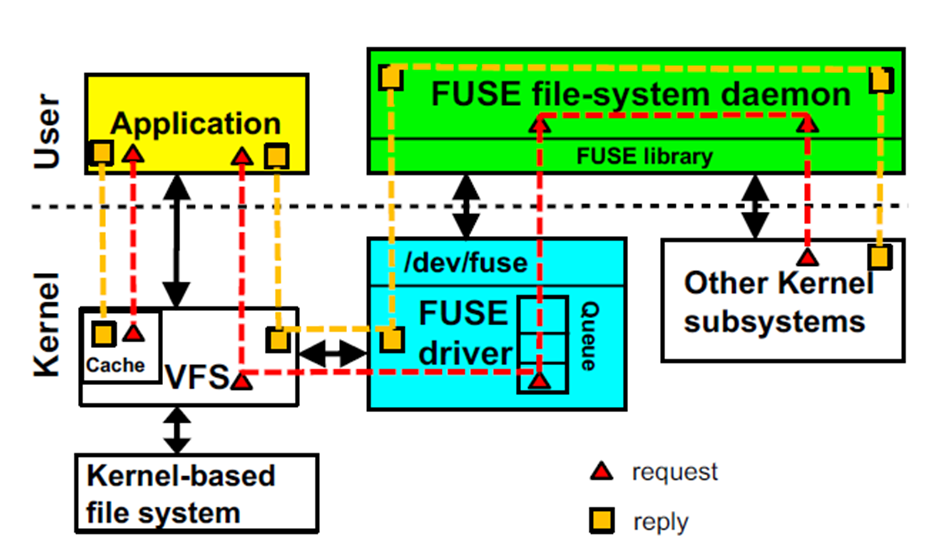
1. Introduction



* traits

1. not a real filesystem, but a process
2. load, register a fuse file-system driver with Linux VFS
3. fuse act as proxy
4. registers a block device /dev/fuse and set up communication between kernel and fuse daemon

* pros

1. the crash of fuse filesystem does not affect the kernel
2. allow user to do more file operation in user space
3. more convenient to develop the filesystem with other applications, such as network-based filesystem
4. allow combination of GPL and non-GPL

* cons

1. even the optimized version have poor performance compared with filesystem in kernel
2. occupy process resource

* Workflow

1. request
   1. User’s application send request as normal file operation.
   2. And then, VFS transmit request by calling the file operation of registered file system.
   3. Finally, /dev/fuse transmit the request to user-level fuse daemon.
2. reply
   1. The reply from fuse daemon is exactly the same flow in the converse way

* Queue

主要有5個queue：

1. interrupts: INTERRUPT requests in the interrupts queue
2. forgets: FORGET requests in the forgets queue
3. pending: synchronous requests (e.g., metadata) in the pending queue.
4. processing: The oldest request in the pending queue is transferred to the user space and simultaneously moved to the processing queue.
5. background: The background queue is for staging asynchronous requests. read requests, write requests (writeback cache enabled)

* API

1. Open
2. userlevel app調用glibc open接口，觸發sys\_open系統調用。
3. sys\_open 調用fuse中inode節點定義的open方法。
4. inode中open生成一個request，並通過/dev/fuse發送request消息到用戶態libfuse。
5. Libfuse調用fuse\_application用戶自定義的open的方法，並將返回值通過/dev/fuse通知給kernel。
6. kernel收到request的處理完成的喚醒，並將結果放回給VFS系統調用結果。
7. userlevel app收到open的reply。
8. Write
9. user在mount目錄下面，對一個regular file調用write, 這一步是在userspace執行
10. write內部會調用VFS提供的一致性接口vfs\_write
11. 根據FUSE module註冊的file\_operations，vfs\_write會調用fuse\_file\_aio\_write，將write request放入fuse connection的request pending queue, 隨後sleep等待fuse daemon reply
12. userspace的libfuse有一個priviledged process函式fuse\_session\_loop polling /dev/fuse, 一旦request queue有請求即通過fuse\_kern\_chan\_receive接收
13. fuse\_kern\_chan\_receive通過read讀取request queue中的內容，read系統調用實際上是調用的device driver接口fuse\_dev\_read
14. 在userspace讀取並分析，執行用戶定義的write操作，將狀態通過fuse\_reply\_write返回給kernel
15. fuse\_reply\_write調用VFS提供的一致性接口vfs\_write
16. vfs\_write最終調用fuse\_dev\_write將執行結果返回給第3步中等待在waitq的process，此process得到reply 後，write返回
17. Mount

FUSE module註冊了fuseblk\_fs\_type和fuse\_fs\_type兩種文件類型。default使用的是fuse\_fs\_type，即mount 函式pointer被初始化為fuse\_mount，而fuse\_mount實際調用mount\_nodev，它主要由以下兩步組成：

* 1. sget(fs\_type)搜索filesystem的super\_block表(type->fs\_supers),如果找到一個與block device相關的super\_block，則返回它的地址。否則，分配並初始化一個新的super\_block，把它插入到filesystem表和global的super\_block表中，並返回其地址。
  2. fill\_super(此函數由各filesystem自行定義)實際上是fuse\_fill\_super。一般而言，fill\_super會分配inode和對應的directory entry，並填充super\_block字段值，另外對於fuse還需要分配fuse\_conn, fuse\_req。此外，它在底層調用了fuse\_init\_file\_inode用fuse\_file\_operations和fuse\_file\_aops分別初始化inode->i\_fop和inode->i\_data.a\_ops。
  3. 在最後的fuse\_fill\_super部分，file就是通過mount傳進來的參數 ”fd=3” 得到的，對應於打開的 ”/dev/fuse” 。在掛載時候創建的super\_block, fuse\_conn, fuse\_dev, file在這裡連接起來。

1. Unlink
   1. fuse daemon會輪詢(polling) /dev/fuse有沒有request，沒有的話會sleep在fc->waitq
   2. 直到sys\_unlink發request從VFS到/dev/fuse
   3. daemon起床後會把requset從pending list刪掉，加入processing list，並且copy到userspace buffer
   4. 接著執行你定義在deamon的operation然後一路傳會application

* Struct
  1. Kernel

1. struct fuse\_conn

每一次mount會宣告一個struct fuse\_conn(即fuse connection)，它代表了userspace和kernelspace的溝通管道。fuse connection維護了包括pending list, processing list和io\_list在內的request queue，fuse connection通過這些list管理userspace和kernelspace之間的溝通。

1. struct fuse\_req

每次執行系統調用時會生成一個struct fuse\_req, 這些fuse\_req依據state被放在不同的list中，struct fuse\_conn維護了這些list。

1. struct file

存放open file與process之間進行互動的資訊，描述了process怎樣與一個open file互動，這種資訊僅存在於當process open file期間存在於kernelspace的memory中。

1. struct inode

filesystem處理文件所需要得所有資訊都放在一個名為inode(索引節點)的資料結構中。filename可以隨時更改，但是file對應的inode是唯一的，並且隨着file的存在而存在

1. struct file\_operation

定義了可以對file執行的操作。

* 1. User
     1. struct fuse\_req

這個結構和kernel的fuse\_req同名，有類似的作用，但是成員不同。

* + 1. struct fuse\_session

定義了user端管理session的struct，包含了一組對session可以執行的操作。

* + 1. struct fuse\_chan

定義了user端與FUSE kernel連接的struct，包含了一組對channel可以執行的操作。

* + 1. struct fuse\_ll\_ops

struct的成員為一個函式func和字串name，kernel發過來的每一個request最後都映射到以此struct為元素的array中。

1. Q&A
   * Fuse 會浪費多層的filesystem interface，有帶來什麼好處?
     1. 上層(user-space)可能有更多發展，network的應用或加密的應用
     2. 下層(kernel-space)確實會浪費不少資源，但是可以方便直接在 userspace 開發 filesystem，kernel 也比較不會被汙染
   * FUSE daemon 會怎麼polling requests? /dev/fuse 怎麼 wake up daemon?
     1. Fuse daemon會透過fuse\_session\_loop polling /dev/fuse
     2. 透過sleep進入wait\_q的process可以透過request和reply叫醒