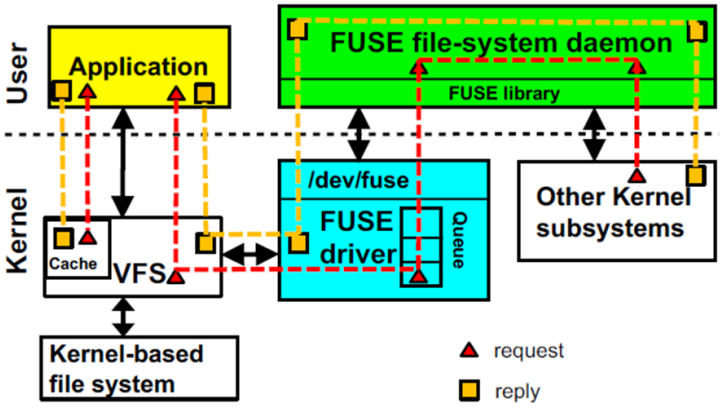
I. Introduction



- 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
     i. User's application send request as normal file operation.
     ii. And then, VFS transmit request by calling the file operation of registered file system.
     iii. Finally, /dev/fuse transmit the request to user-level fuse daemon.
  2. reply
     i. 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
   i. userlevel app 調用 glibc open 接口，觸發 sys_open 系統調用。
   ii. sys_open 調用 fuse 中 inode 節點定義的 open 方法。
   iii. inode 中 open 生成一個 request，並通過/dev/fuse 發送 request 消息到用戶態 libfuse。
   iv. 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。
   v. 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。
   vi. userlevel app 收到 open 的 reply。
2. Write
   i. user 在 mount 目錄下面，對一個 regular file 調用 write, 這一步是在 userspace 執行
   ii. write 內部會調用 VFS 提供的一致性接口 vfs_write
   iii. 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
   iv. 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
   v. 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
   vi. 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

vii. fuse_reply_write 調用 VFS 提供的一致性接口 vfs_write
viii. 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

3. 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：
i. 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。
ii. 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。
iii. 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。

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

- Struct

1. Kernel
i. 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。

ii. 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。

iii. 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。

iv. 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

v. struct file_operation
定義了可以對 file 執行的操作。

2. User

i. 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。

ii. 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。

iii. 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。

iv. 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。

II. 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