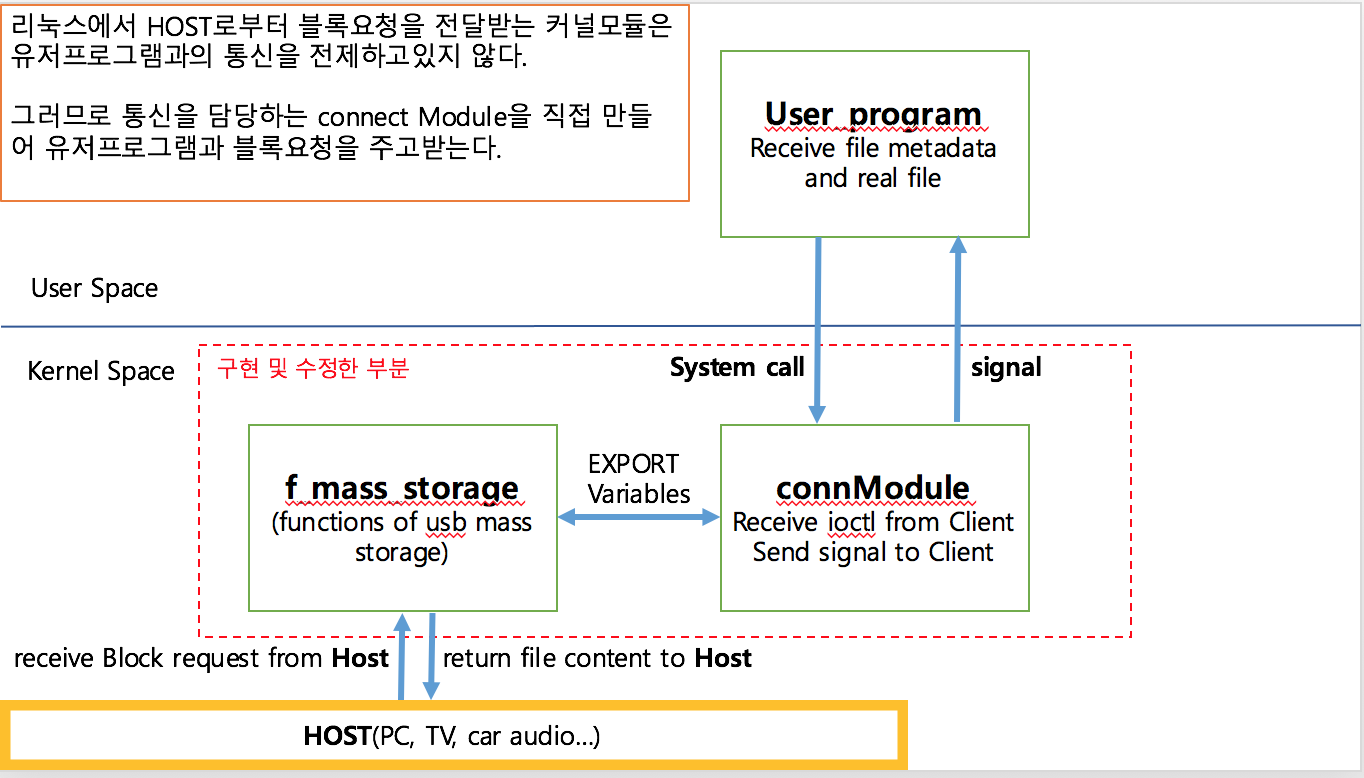
**구름USB 커널공간 연결모듈**

작성자 이창현



1. **연결 모듈의 역할**

구름USB 내부에서는 리눅스 운영체제가 작동하고, 리눅스의 가젯 스토리지를 이용해 HOST(TV, PC등)에 USB로서 연결됩니다. 구름USB는 가젯 드라이버 중 g\_mass\_storage와 usb\_f\_mass\_storage를 이용하며, g\_mass\_storage는 구름USB가 HOST에 USB로서 연결되도록 도와주고, f\_mass\_storage는 가젯드라이버 내부의 기능을 탑재하고 있습니다. 구름 USB는 g\_mass\_storage는 그대로 이용하지만, f\_mass\_storage 모듈은 수정하여 사용합니다. 기존의 f\_mass\_storage는 USB 내부의 FAT이미지파일을 읽도록 되어있지만, 우리가 원하는것은 USB 외부의 클라우드 저장공간의 파일을 읽어오는 것입니다. 클라우드 저장공간의 파일을 읽어오는 역할은 유저공간의 프로그램이 담당하고 있으므로 해야할 일은 f\_mass\_storage의 실행흐름을 가로채 HOST의 block 요청을 그대로 유저프로그램에 전달하는 일 뿐입니다.

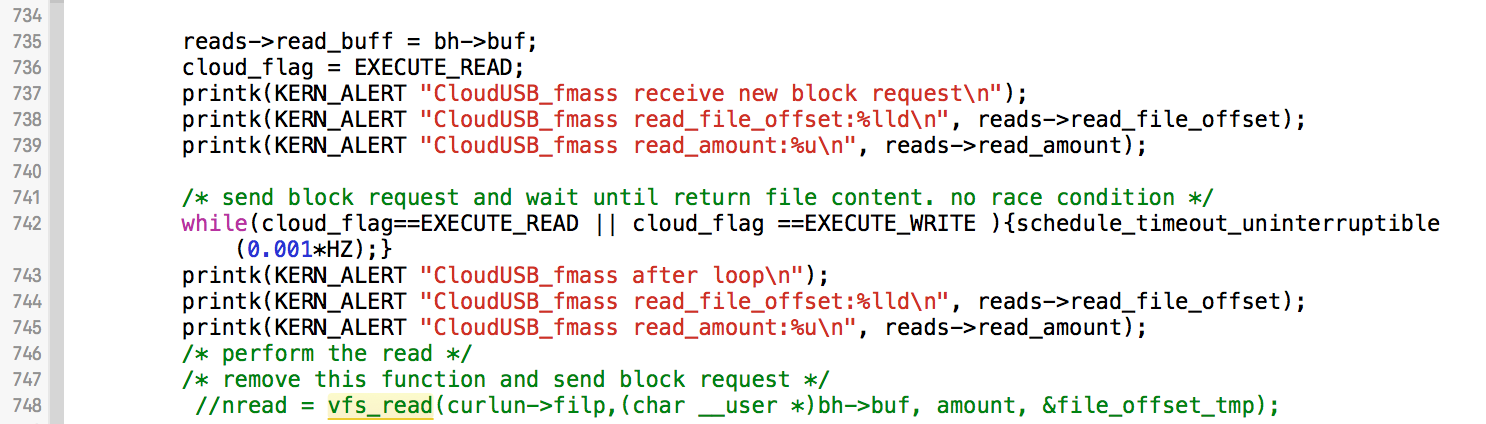
1. **Block 요청**

Block 요청은 HOST가 구름USB내부의 특정 파일을 가져다 달라고 요청하거나(READ) 특정 파일을 수정해달라는 요청(WRITE)을 뜻합니다. READ요청은 offset(이미지파일에서 bite단위 시작점)과 amount(읽어올 bite 수)로 이뤄져 있으며, write요청은 offset, amount, buff(써줄 파일내용)으로 구성되어 있습니다. HOST의 요청은 가젯드라이버 소스 중 f\_mass\_storage.c의 do\_read()와 do\_write()에 전달됩니다. 이 함수는 원래 USB내의 이미지파일을 읽어오려고 하지만, connModule로 block 요청을 전달하도록 수정되었습니다.

1. **connModule의 원리**

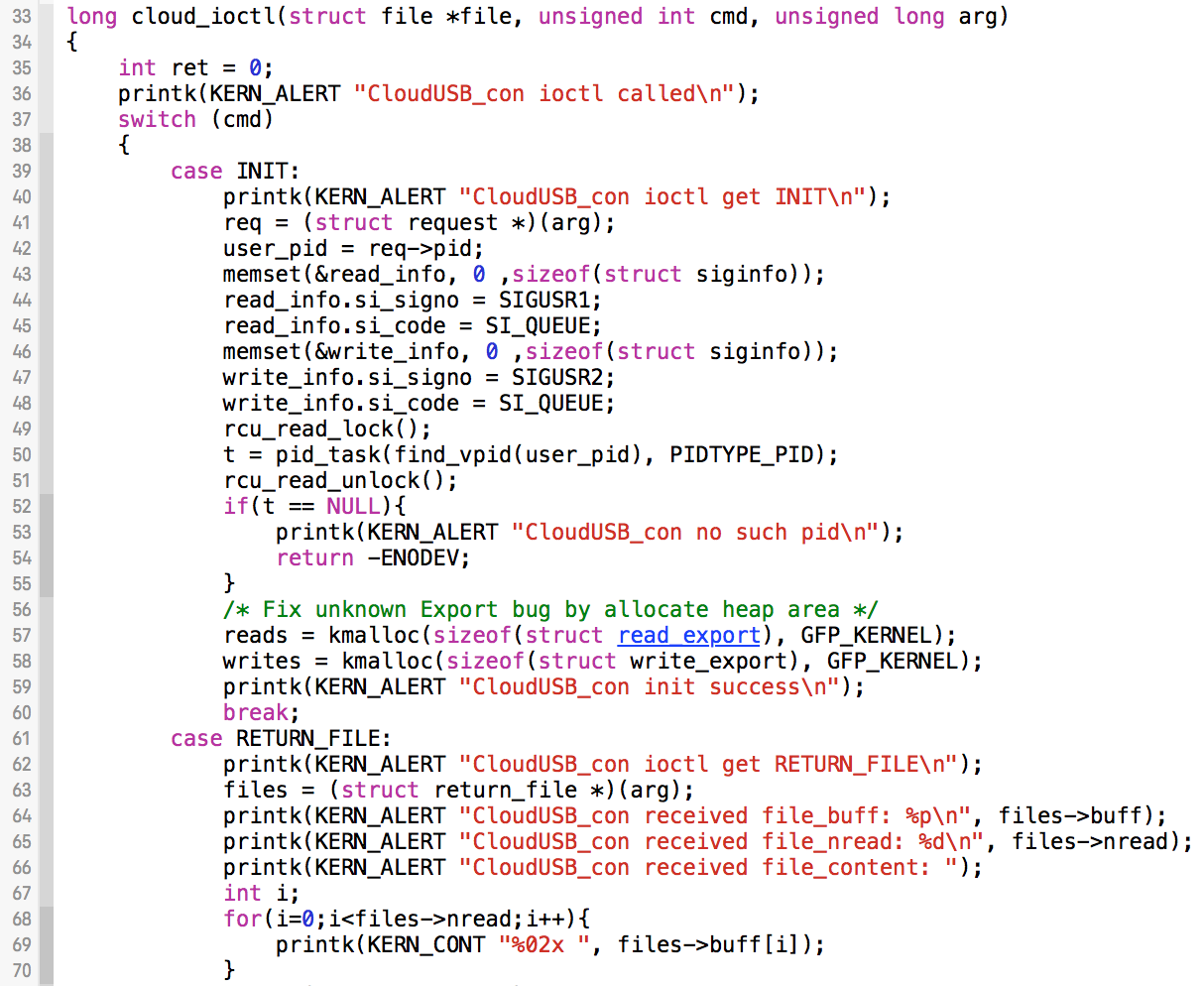
connModule은 유저 프로그램에 block 요청을 전달하고 그 결과를 리턴받는 커널 모듈입니다. connModule은 symbol export로 block요청에 대한 구조체를 f\_mass\_storage로부터 넘겨받습니다. f\_mass\_storage가 직접 유저프로그램과 통신하지 않고 connModule을 따로 이용하는 이유는 세가지가 있습니다. 첫번째로 3천여줄이 넘는 f\_mass\_storage.c에 새로운 코드를 추가하고 디버깅하는것이 불편하고, 두번째로 ioctl systemcall을 이용하기 위해 usb\_f\_mass\_storage커널모듈의 init함수에 코드를 추가하는것이 힘들고, 세번째로 커널단에서 모든 메모리는 서로 공유되기 때문에 symbol export로 인한 오버헤드가 경미하다고 생각했기 때문입니다. connModule은 유저 프로그램에 block 요청을 전달하기 위해 signal을 이용하고, 유저 프로그램은 connModule에 파일정보 등을 리턴하기 위해 ioctl systemcall을 이용합니다. 하지만 signal은 말그대로 신호를 전달하는것 뿐 block 요청을 담을수는 없습니다. 그래서 connModule이 적재된 후 유저프로그램이 실행될 때 자동으로 init단계가 실행되며, 이때 ioctl을 통해 connModule이 block 요청을 써줄 구조체를 동적할당하고 이 구조체의 선두주소를 ioctl로 connModule에 전달합니다. connModule은 이후 유저프로그램에 block요청을 보내야 할 때 전달받은 구조체주소에 block 요청을 써준 후 signal을 보내며, signal받은 유저프로그램은 해당 구조체에서 block 요청을 읽어들입니다.

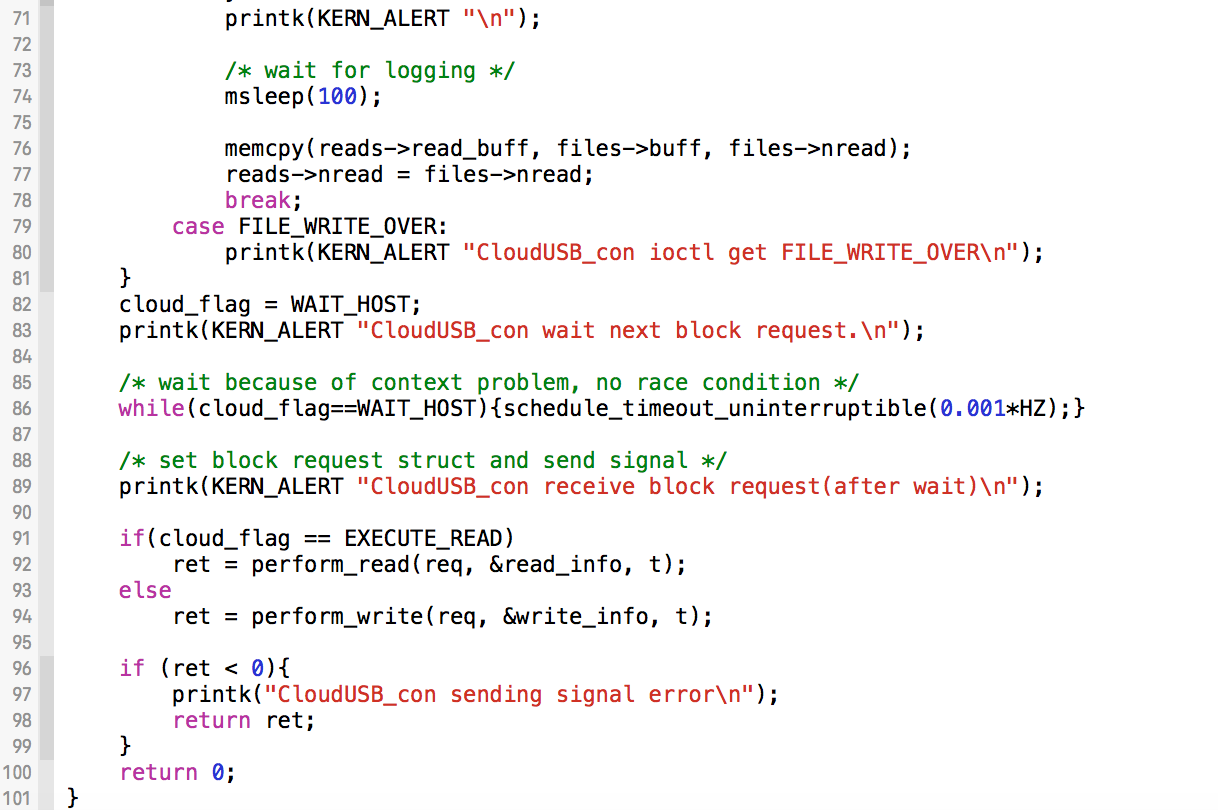
1. **간략한 코드 설명**



f\_mass\_storage.c 기존의 do\_read()함수에서 수정된 부분

748줄의 vfs\_read()함수는 라즈베리파이 내의 FAT이미지파일에서 정보를 읽어오기 위해 존재하던 함수였습니다. 이 코드를 주석처리하고 flag를 이용해 connModule로 실행흐름을 전달하는 제어문을 넣었습니다. HOST가 동시에 여러개의 block요청을 보내는 일은 없으므로 이렇게 단순한 실행흐름 제어방법을 이용해도 문제없습니다.





connModule.c의 cloud\_ioctl()함수

connModule에 새로운 block요청이 들어오기 전에는 86줄의 while문을 돌며 대기하고 있습니다. flag제어를 통해 Block요청이 도달하면 while문을 벗어나고 signal으로 유저프로그램에게 block요청을 전달합니다. 유저프로그램이 요청에 대한 작업을 종료하면 실행흐름은 다시 connModule로 돌아와 while문에서 대기하게 됩니다. 과정이 다소 지저분하지만 유저프로그램이 전달해준 구조체주소에 블록요청을 써주기 위해선 유저프로그램의 context가 유지되어야 하므로 위와 같은 방법을 선택했습니다.

1. **주의사항 등**

- connModule.h의 MODULE\_LICENSE(“GPL”)이 없으면 connModule.c의 pid\_task()가 작동하지 않습니다. 향후 비공개소스로 전환할 예정이라면 대체함수를 찾아야 합니다.

- 현재 mac os 이외에 연결할때에는 제대로 작동하지 않습니다.

실제파일을 읽을 때 어떻게 작동하는지 f\_mass\_storage.c의 do\_scsi\_command() 등에서 로그를 찍어보는것이 도움이 될 수 있습니다.

- 모듈 적재와 프로그램 실행 순서를 지켜야 합니다(connModule -> 유저프로그램 -> usb\_f\_mass\_storage -> g\_mass\_storage). 또한 connModule과 유저프로그램간의 init과정이 끝난 후 g\_mass\_storage모듈을 적재해야 합니다.

- 각종 에러발생시 dmesg 또는 grep -r "CloudUSB" /var/log 를 참고하면 됩니다. 만약 아무이유없이 모듈적재가 안되면 connModule을 clean하고 다시 make하면 해결될 수 있습니다.

- f\_mass\_storage.c와 connModule의 busy-waiting을 wait queue로 대체하면 약간의 성능향상을 기대할 수 있습니다.