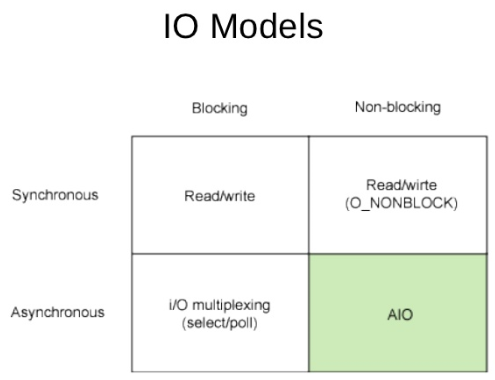
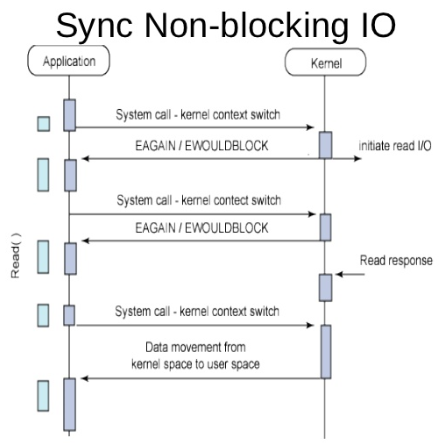
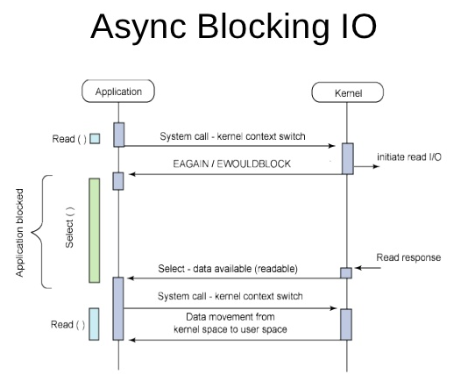
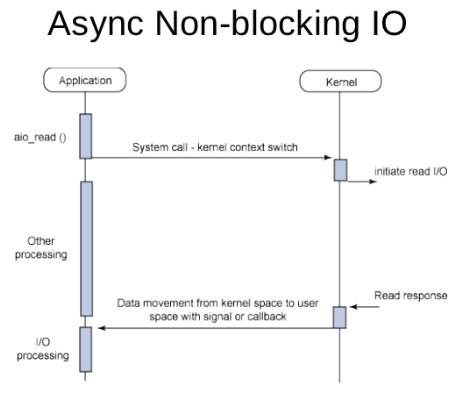
**POSIX Asynchronous IO**

# POSIX IO Model









Linux에서 Async Non-blocking IO에는 POSIX AIO & Linux native aio 두 가지 종류가 있다.

# POSIX Realtime Signal

http://www.joinc.co.kr/modules/moniwiki/wiki.php/Site/Network\_Programing/AdvancedComm/RTS1 -> Realtime signal에 대한 설명, 보통 시그널과의 차이점및 간단한 예제

What I seem to understand:

POSIX AIO APIs are prototyped in <aio.h> and you link your program with librt(-lrt), while the libaio APIs in <libaio.h> and your program is linked with libaio (-laio).

What I can't figure out:

1.Does the kernel handle the either of these methods differently?

2.Is the O\_DIRECT flag mandatory for using either of them?

As mentioned in [this post](http://stackoverflow.com/questions/6918530/linux-kernel-aio-functionality), libaio works fine without O\_DIRECT when using libaio.Okay,understood but:

According to R.Love's Linux System Programming book, Linux supports aio (which I assume is POSIX AIO) on regular files only if opened with O\_DIRECT.But a small program that I wrote (using aio.h,linked with -lrt) that calls aio\_write on a file opened without the O\_DIRECT flag works without issues.

==>

On linux, the two AIO implementations are fundamentally different.

The POSIX AIO is a user-level implementation that performs normal blocking I/O in multiple threads, hence giving the illusion that the I/Os are asynchronous. The main reason to do this is that:

1. it works with any filesystem
2. it works (essentially) on any operating system (keep in mind that gnu's libc is portable)
3. it works on files with buffering enabled (i.e. no O\_DIRECT flag set)

The main drawback is that your queue depth (i.e. the number of outstanding operations you can have in practice) is limited by the number of threads you choose to have, which also means that a slow operation on one disk may block an operation going to a different disk. It also affects which I/Os (or how many) is seen by the kernel and the disk scheduler as well.

The kernel AIO (i.e. io\_submit() et.al.) is kernel support for asynchronous I/O operations, where the io requests are actually queued up in the kernel, sorted by whatever disk scheduler you have, presumably some of them are forwarded (in somewhat optimal order one would hope) to the actual disk as asynchronous operations (using TCQ or NCQ). The main restriction with this approach is that not all filesystems work that well or at all with async I/O (and may fall back to blocking semantics), files have to be opened with O\_DIRECT which comes with a whole lot of other restrictions on the I/O requests. If you fail to open your files with O\_DIRECT, it may still "work", as in you get the right data back, but it probably isn't done asynchronously, but is falling back to blocking semantics.

Also keep in mind that io\_submit() can actually block on the disk under certain circumstances.