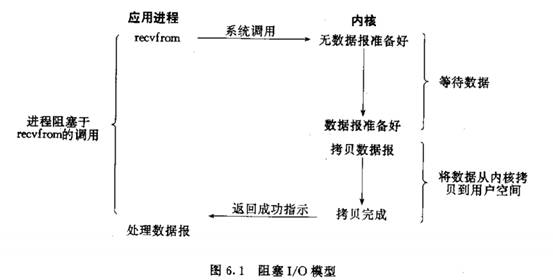
IO基础入门之Linux网络I/O模型简介

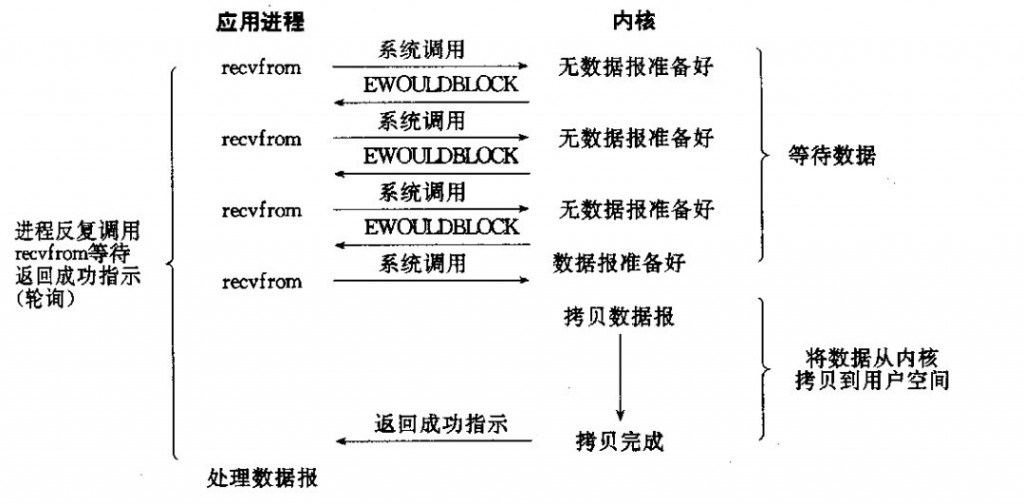
Linux内核将所有的外部设备都看作一个文件来操作，对一个文件的读写操作会调用内核提供的系统命令，返回一个file descriptor(fd，文件描述符)。而对一个socket的读写也会有相应的描述符，称为socketfd（scoket 描述符），描述符就是一个数字，它指向内核中的一个结构体（文件路径，数据区等一些属性）。

根据UNIX网络编程对I/O模型的分类，UNIX提供了5种I/O模型，分别如下：

1 阻塞I/O模型:最常用的I/O模型就是阻塞I/O模型，缺省的情形下，所有文件的操作都是阻塞的。我们以套接字接口为例来讲解此模型。再进程空间中调用recvform，其系统调用直到数据包到达且被复制到应用进程的缓冲区中或者发生错误的时候才返回。在此期间一直等待，进程在从调用recvform开始到它返回的整段时间内是被阻塞的，因此被称为阻塞I/O模型。

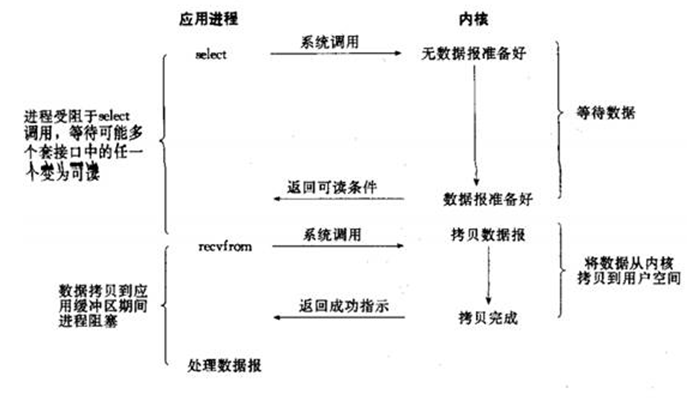


2. 非阻塞I/O模型：rcvform从应用层到内核的时候，如果该缓冲区没有数据的话，就直接返回一个EWOULDBLOCK错误，一般都对非阻塞I/O模型进行轮询检查这个状态，看内核是不是有数据到来。

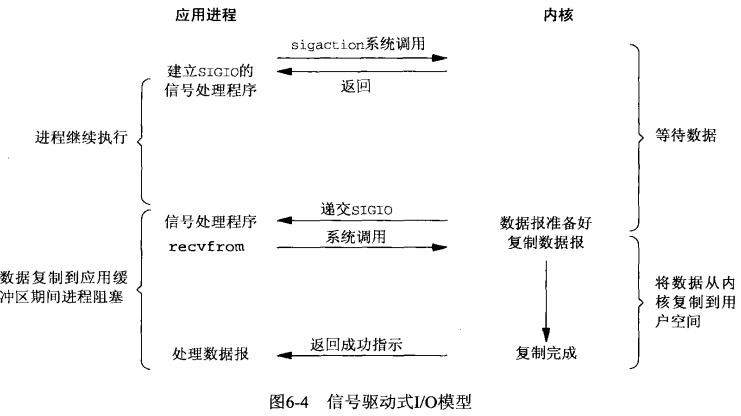


3 I/O复用模型: Linux提供select/poll（I/O复用模型会用到selsect或者poll函数，这两个函数也会是进程阻塞，但是和阻塞I/O所不同的是，这两个函数可以同时阻塞多个I/O操作。而且可以同时对多个读操作，多个写操作的I/O函数进行检测，直到有数据可读或可写，才真正调用I/O操作函数），进程通过将一个或多个fd传递给select或poll系统调用，阻塞在select操作上，这样selsect/poll可以帮我们侦测多fd是否处于就绪状态。Select/poll是顺序扫描fd是否就绪，而且支持的fd数量有限，因此它的使用受到了一些制约。

Linux还提供了一个epoll系统调用，epoll使用基于事件驱动方式代替顺序扫描，因此性能更高。当有fd就绪时，立即回掉函数rollback。



4 信号驱动I/O模型： 首先开启套接口信号驱动I/O功能，并通过系统调用digaction执行一个信号处理函数（此系统调用立即返回，进程继续工作，它是非阻塞的）。当数据准备就绪时，就为该进程生成一个SIGIO信号，通过信号回调通知应用程序调用recvform来读取数据，并通知主循环函数处理数据。



5 异步I/O：告知内核启动某个操作，并让内核在整个操作完成之后（包括将数据从内核复制到用户自己的缓冲区）通知我们。这种模型与信号驱动模型的主要区别是：信号驱动I/O由内核通知我们合适可以开始一个I/O操作；异步I/O模型由内核通知我们I/O操作合适已经完成。

