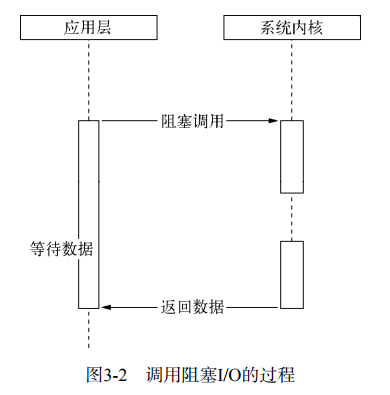
# 异步I/O（pdf65页）

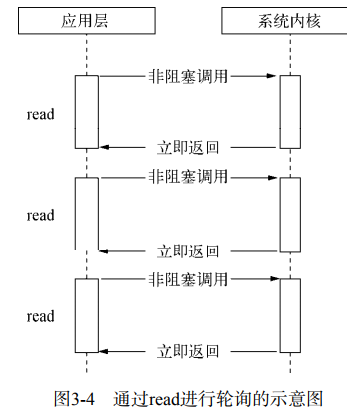
3.2.1阻塞IO和非阻塞IO



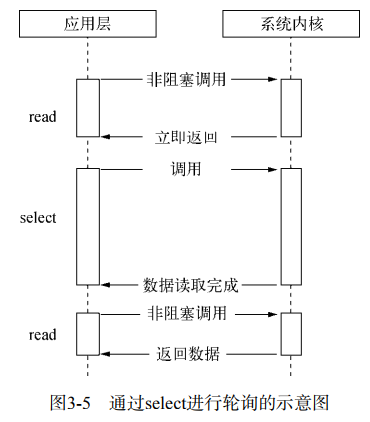
阻塞IO需要等待获取数据过程全部完成才返回，非阻塞IO则立即返回，但是时刻轮询获取数据是否完成。阻塞IO造成CPU等待浪费，非阻塞IO造成CPU资源浪费。

现存的轮询技术主要有：

Read：最原始，性能最低。

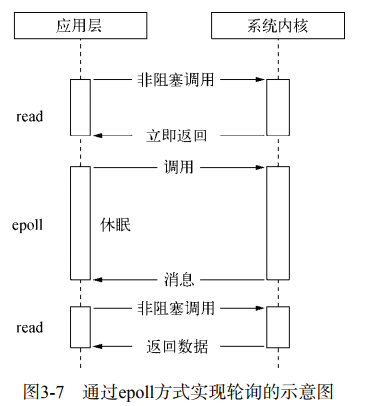


Select：通过文件描述符上的事件状态来进行判断。它用1024长度数组来保存文件状态，即最多同时检查1024个文件的状态。



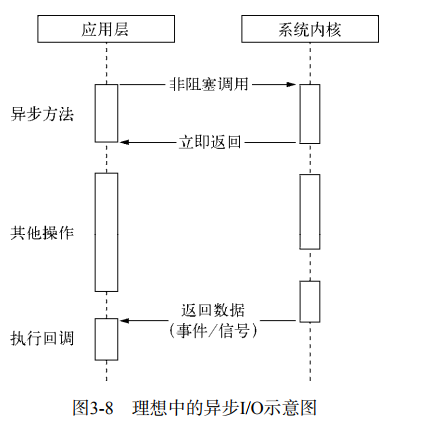
Poll：类似select，不过用采用链表的方式避免数组长度和不需要的检查。

Epoll：linux下最高效IO，进入轮询后没有IO事件则休眠，直到事件回调，这是一个正真利用了事件回调方式的轮询。不会浪费CPU性能。

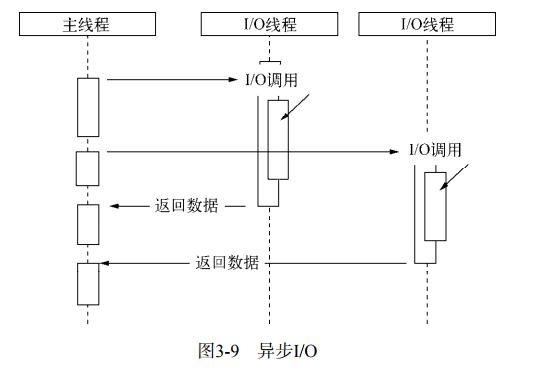


3.2.2 理想的非阻塞异步IO

尽管epoll已经利用事件来降低CPU消耗率，但是休眠期间CPU几乎闲置。理想的轮询应该是



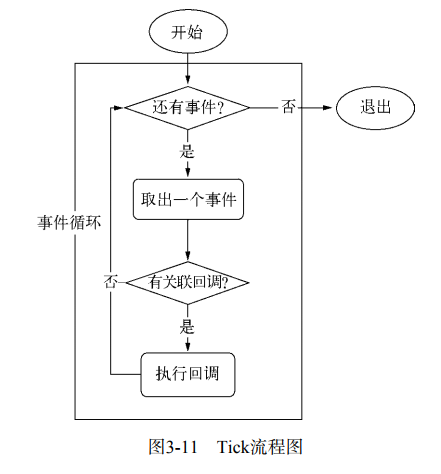
3.2.3 现实的异步IO



3.3 node的异步IO

3.3.1 事件循环

进程启动时，node会开一个while（1）的死循环，每次循环称之为TIck。每次Tick就检查是否是事件回调，如果就就执行回调。

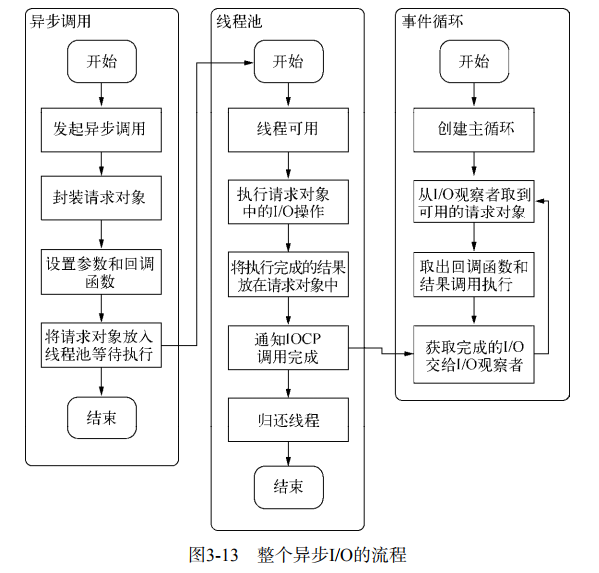


3.3.2 观察者

每次Tick的过程都会有一个或多个观察者，判断是否有事件要处理就是询问观察者是否有事件要处理的过程。

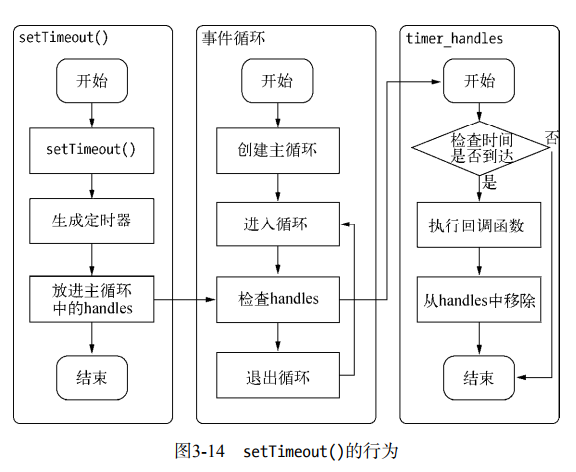
在node中，事件主要来源是网络请求和文件IO。

3.3.3 请求对象



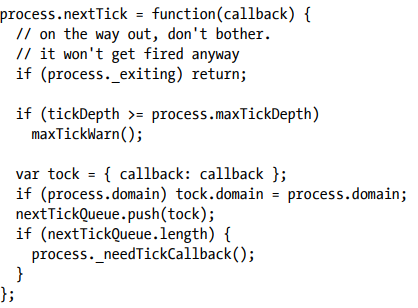
3.4 非IO的异步回调

3.4.1 定时器



3.4.2 process.nextTick

process.nextTick相较于定时器比较轻量。



3.4.3 setImmediate

与nextTIck类似，不过setImmediate要比nextTick执行慢。nextTIck属于idle观察者，setImmediate属于check观察者，优先级为 idle观察者 > IO观察者 > check观察者。

并且process.nextTick会把回调保存在一个数组里，每次循环都会将数组里的函数执行完，而setImmediate的回调保存在链表中，每次循环只执行一个回调。[Demo](demo.js)（没有实现预期效果）