

我读过的最好的epoll讲解--转自“知乎”

首先我们来定义流的概念，一个流可以是文件，socket，pipe等等可以进行I/O操作的内核对象。

不管是文件，还是套接字，还是管道，我们都可以把他们看作流。

之后我们来讨论I/O的操作，通过read，我们可以从流中读入数据；通过write，我们可以往流写入数据。现在假定一个情形，我们需要从流中读数据，但是流中还没有数据，（典型的例子为，客户端要从socket读如数据，但是服务器还没有把数据传回来），这时候该怎么办？

阻塞：阻塞是个什么概念呢？比如某个时候你在等快递，但是你不知道快递什么时候过来，而且你没有别的事可以干（或者说接下来的事要等快递来了才能做）；那么你可以去睡觉了，因为你知道快递把货送来时一定会给你打个电话（假定一定能叫醒你）。

非阻塞忙轮询：接着上面等快递的例子，如果用忙轮询的方法，那么你需要知道快递员的手机号，然后每分钟给他挂个电话：“你到了没？”

很明显一般人不会用第二种做法，不仅显很无脑，浪费话费不说，还占用了快递员大量的时间。

大部分程序也不会用第二种做法，因为第一种方法经济而简单，经济是指消耗很少的CPU时间，如果线程睡眠了，就掉出了系统的调度队列，暂时不会去瓜分CPU宝贵的时间片了。

为了了解阻塞是如何进行的，我们来讨论缓冲区，以及内核缓冲区，最终把I/O事件解释清楚。缓冲区的引入是为了减少频繁I/O操作而引起频繁的系统调用（你知道它很慢的），当你操作一个流时，更多的是以缓冲区为单位进行操作，这是相对于用户空间而言。对于内核来说，也需要缓冲区。

假设有一个管道，进程A为管道的写入方，B为管道的读出方。

假设一开始内核缓冲区是空的，B作为读出方，被阻塞着。然后首先A往管道写入，这时候内核缓冲区由空的状态变到非空状态，内核就会产生一个事件告诉B该醒来了，这个事件姑且称之为“缓冲区非空”。

但是“缓冲区非空”事件通知B后，B却还没有读出数据；且内核许诺了不能把写入管道中的数据丢掉这个时候，A写入的数据会滞留在内核缓冲区中，如果内核也缓冲区满了，B仍未开始读数据，最终内核缓冲区会被填满，这个时候会产生一个I/O事件，告诉进程A，你该等等（阻塞）了，我们把这个事件定义为“缓冲区满”。

假设后来B终于开始读数据了，于是内核的缓冲区空了出来，这时候内核会告诉A，内核缓冲区有空位了，你可以从长眠中醒来了，继续写数据了，我们把这个事件叫做“缓冲区非满”

也许事件Y1已经通知了A，但是A也没有数据写入了，而B继续读出数据，知道内核缓冲区空了。这个时候内核就告诉B，你需要阻塞了！，我们把这个时间定为“缓冲区空”。

这四个情形涵盖了四个I/O事件，缓冲区满，缓冲区空，缓冲区非空，缓冲区非满（注都是说的内核缓冲区，且这四个术语都是我生造的，仅为解释其原理而造）。这四个I/O事件是进行阻塞同步的根本。（如果不能理解“同步”是什么概念，请学习操作系统的锁，信号量，条件变量等任务同步方面的相关知识）。

然后我们来说一说阻塞I/O的缺点。但是阻塞I/O模式下，一个线程只能处理一个流的I/O事件。如果想要同时处理多个流，要么多进程(fork)，要么多线程(pthread_create)，很不幸这两种方法效率都不高。

于是再来考虑非阻塞忙轮询的I/O方式，我们发现我们可以同时处理多个流了（把一个流从阻塞模式切换到非阻塞模式再此不予讨论）：

我们只要不停的把所有流从头到尾问一遍，又从头开始。这样就可以处理多个流了，但这样的做法显然不好，因为如果所有的流都没有数据，那么只会白白浪费CPU。这里要补充一点，阻塞模式下，内核对于I/O事件的处理是阻塞或者唤醒，而非阻塞模式下则把I/O事件交给其他对象（后文介绍的select以及epoll）处理甚至直接忽略。

为了避免CPU空转，可以引进了一个代理（一开始有一位叫做select的代理，后来又有一位叫做poll的代理，不过两者的本质是一样的）。这个代理比较厉害，可以同时观察许多流的I/O事件，在空闲的时候，会把当前线程阻塞掉，当有一个或多个流有I/O事件时，就从阻塞态中醒来，于是我们的

程序就会轮询一遍所有的流（于是我们可以把“忙”字去掉了）。代码长这样：

于是，如果没有I/O事件产生，我们的程序就会阻塞在select处。但是依然有个问题，我们从select那里仅仅知道了，有I/O事件发生了，但却并不知道是那几个流（可能有一个，多个，甚至全部），我们只能无差别轮询所有流，找出能读出数据，或者写入数据的流，对他们进行操作。

但是使用select，我们有 $O(n)$ 的无差别轮询复杂度，同时处理的流越多，每一次无差别轮询时间就越长。再次

说了这么多，终于能好好解释epoll了

epoll可以理解为event poll，不同于忙轮询和无差别轮询，epoll之会把哪个流发生了怎样的I/O事件通知我们。此时我们对这些流的操作都是有意义的。（复杂度降低到了 $O(1)$ ）

在讨论epoll的实现细节之前，先把epoll的相关操作列出：

epoll_create 创建一个epoll对象，一般`epollfd = epoll_create()`

epoll_ctl（epoll_add/epoll_del的合体），往epoll对象中增加/删除某一个流的某一个事件

比如

`epoll_ctl(epollfd, EPOLL_CTL_ADD, socket, EPOLLIN);`//注册缓冲区非空事件，即有数据流入

`epoll_ctl(epollfd, EPOLL_CTL_DEL, socket, EPOLLOUT);`//注册缓冲区非满事件，即流可以被写入

`epoll_wait(epollfd,...)`等待直到注册的事件发生

（注：当对一个非阻塞流的读写发生缓冲区满或缓冲区空，write/read会返回-1，并设置`errno=EAGAIN`。而epoll只关心缓冲区非满和缓冲区非空事件）。

一个epoll模式的代码大概的样子是：

限于篇幅，我只说这么多，以揭示原理性的东西，至于epoll的使用细节，请参考man和google，实现细节，请参阅linux kernel source。