基础概念

用户空间和内核空间

操作系统的程序是分成系统内核和普通应用程序的。为了保护内核的安全，操作系统将内存分成两部分。

一部分是**内核空间**，一部分是**用户空间**

内核空间和用户空间的寻址是互不干涉的

进程切换

内核具有挂起正在CPU上运行的进程，和恢复某个已经挂起的进程的执行的能力。

进程的挂起和恢复运行叫做**进程切换**。

进程切换是非常消耗资源的。

进程阻塞

当正在执行的进程，由于期待的事件没有发生，比如请求的资源失败，或者是等待某种操作的完成，这种情况下会有操作系统执行阻塞原语(Block)，使得自己的运行状态变成阻塞状态。

由于处于运行状态的进程才会进入阻塞状态。

当进程进入阻塞状态时，是不会占用CPU资源的。

缓存 I/O

缓存IO又叫做标准IO。

在Linux系统中，IO读出来的数据会先缓存在文件系统中的page cache中。

也就是说IO数据会先拷贝到操作系统内核的缓冲区中，然后再被拷贝到用户空间。

I/O执行的两大阶段

在linux操作系统中，IO的读取是不会直接拷贝到应用程序的缓冲区的，而是包括两个阶段。

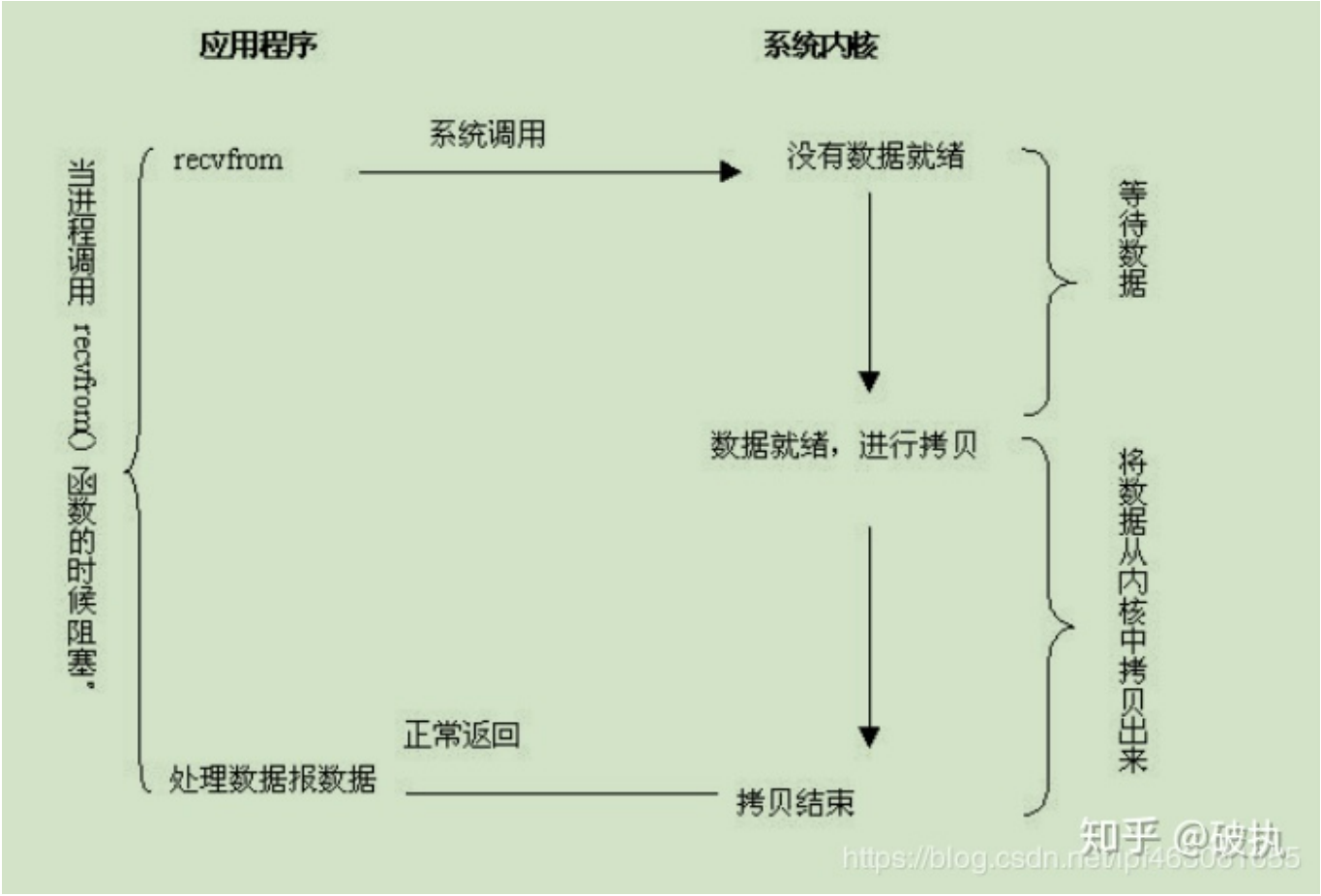
1. 等待数据，复制到内核空间
2. 从内核空间复制数据到用户空间

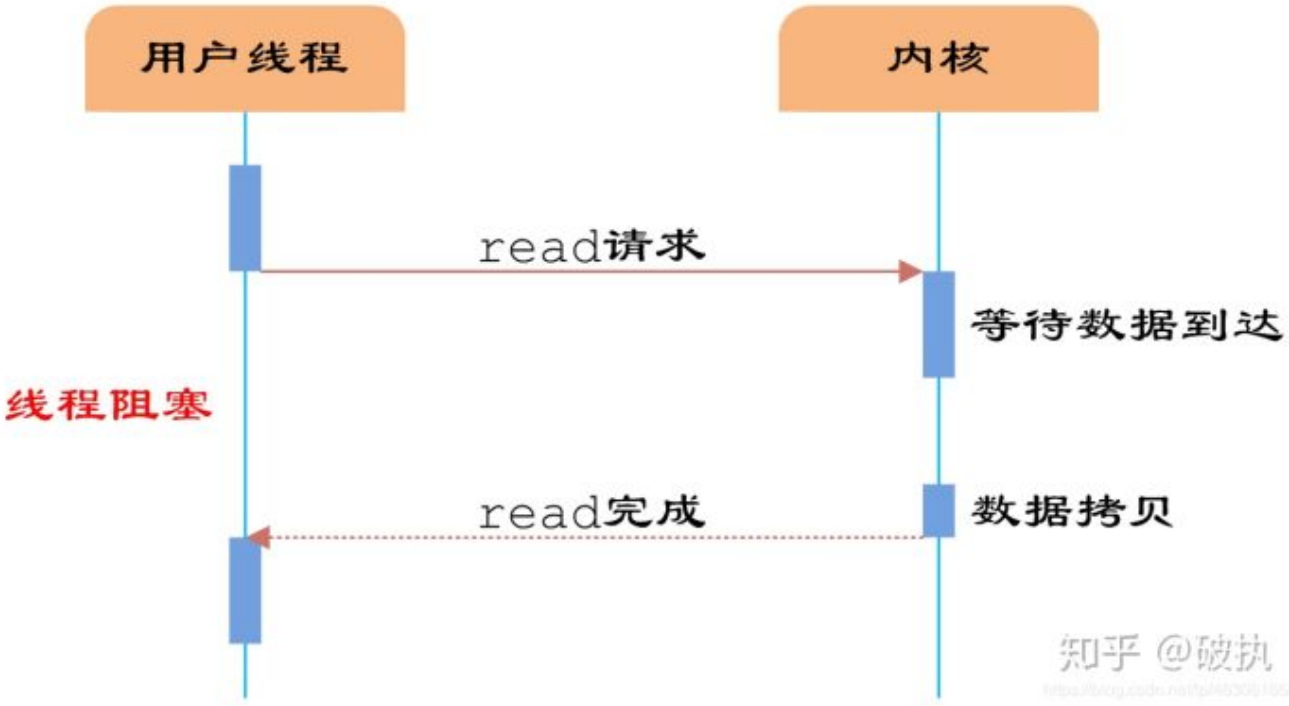
五大IO模型及对比

- 同步阻塞 （bloking IO）
- 同步非阻塞 (nonbloking IO)
- IO多路复用 (IO multiplexing)
- 信号驱动IO (signal driven IO)

- 异步IO (async IO)

同步阻塞（bloking IO）





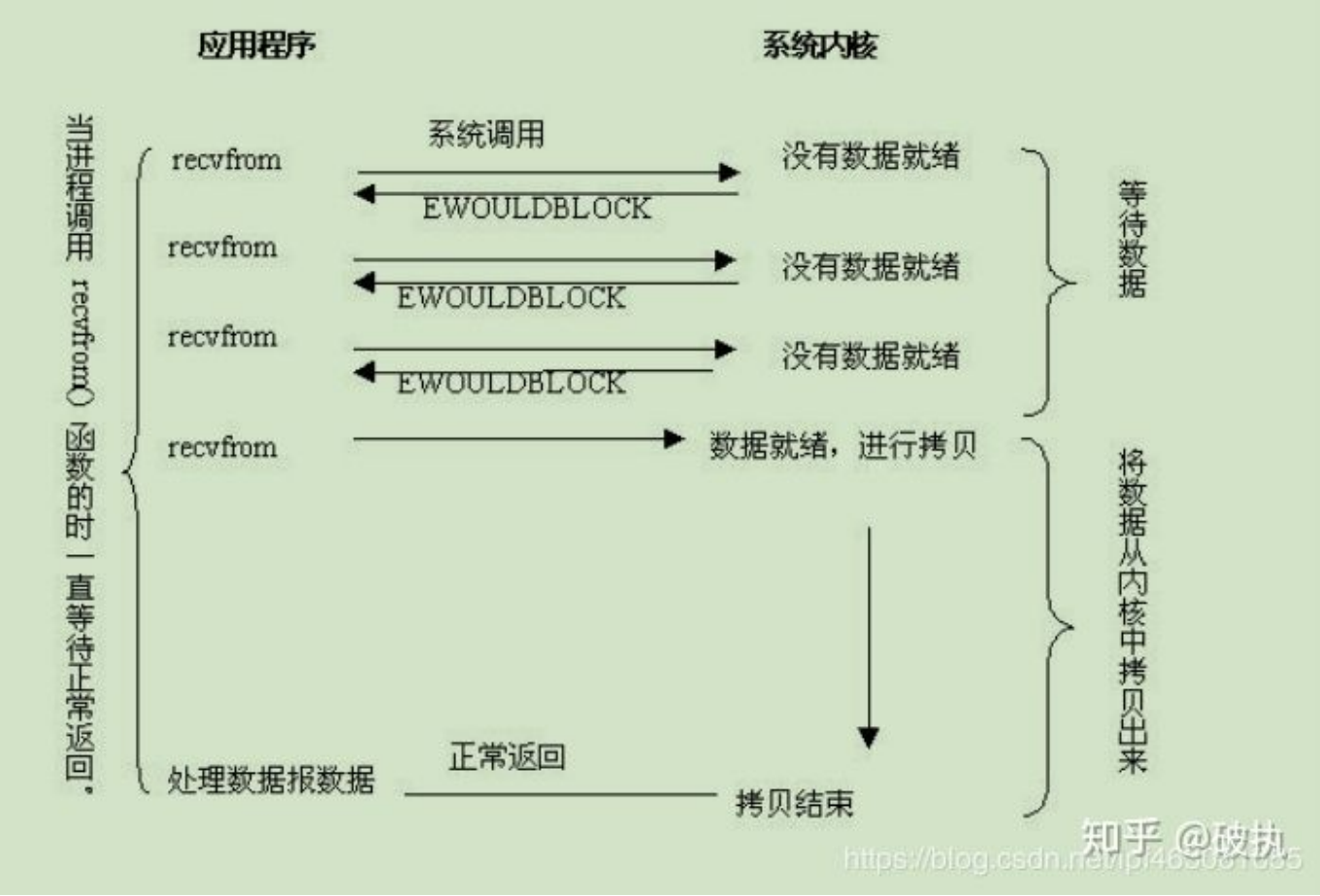
用户线程向内核发出IO请求，内核在等待数据输入，或者正在检索数据，等待数据就绪，然后再进行数据拷贝。

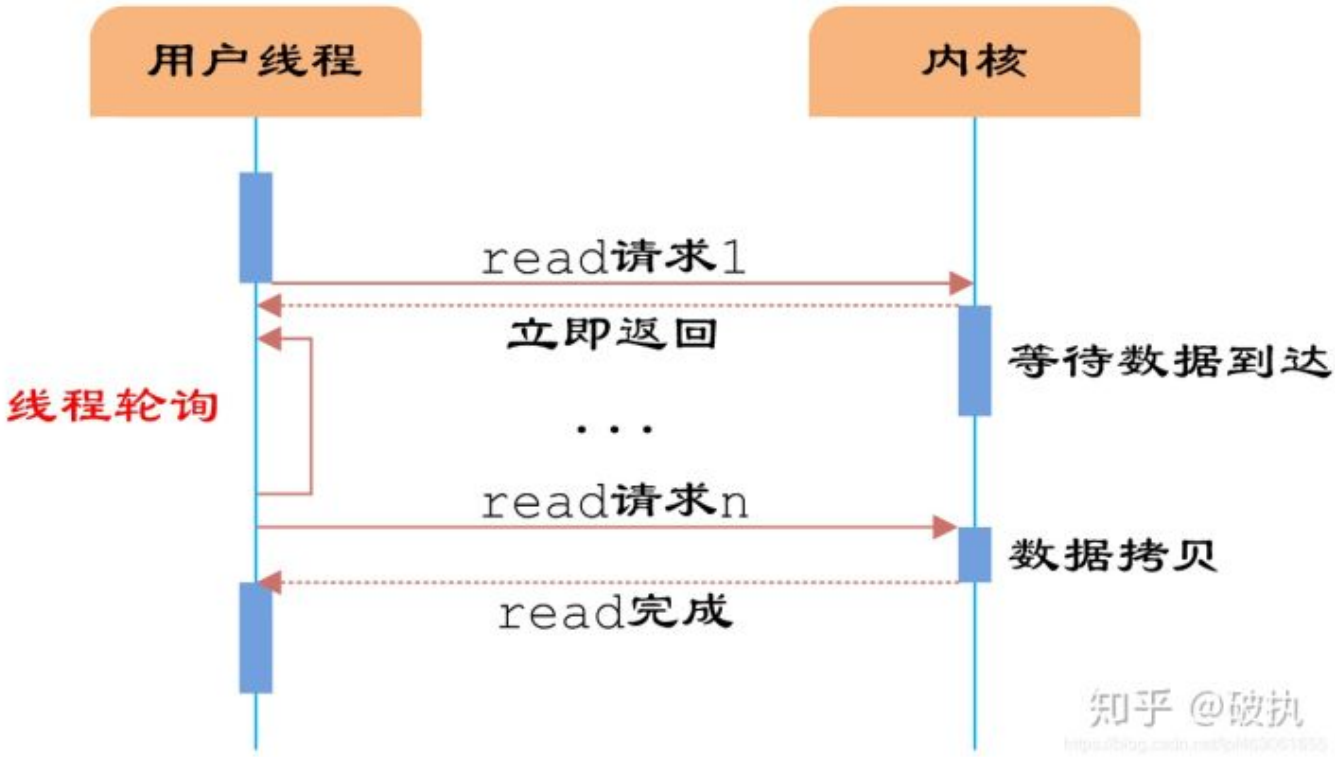
先拷贝进内核空间，然后再把数据从内核空间拷贝进用户空间。

整个IO请求的过程中，用户线程是被阻塞的，这导致用户在发起IO请求时，不能做任何事情，对CPU的资源利用率不够。

同步非阻塞 (nonbloking IO)

同步非阻塞是在同步阻塞IO的基础上，将`socket`设置成`NONBLOCK`模式，就可以变成非阻塞模式了。在此模式下，用户线程发起IO请求之后会立即返回，不会等待阻塞。



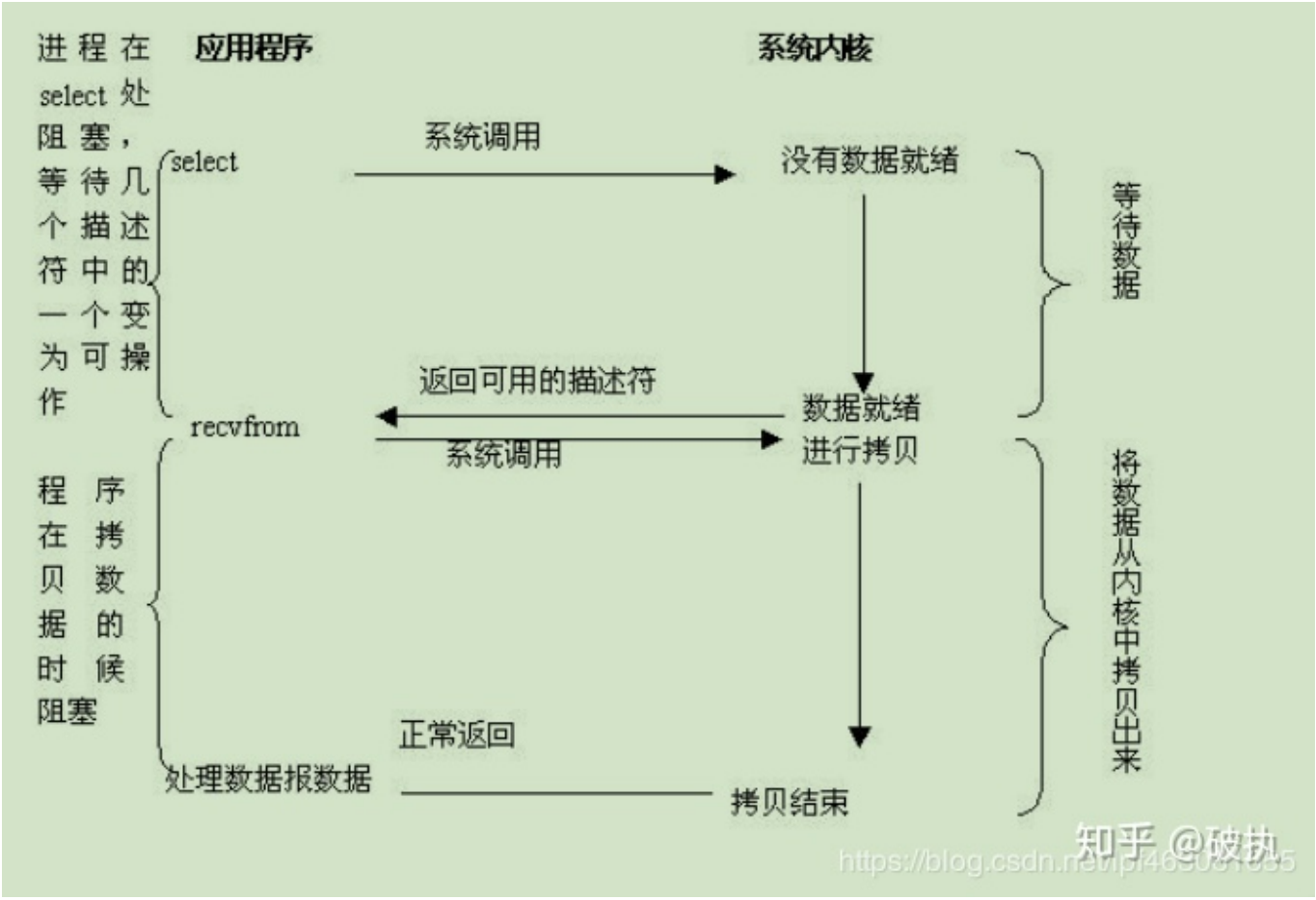


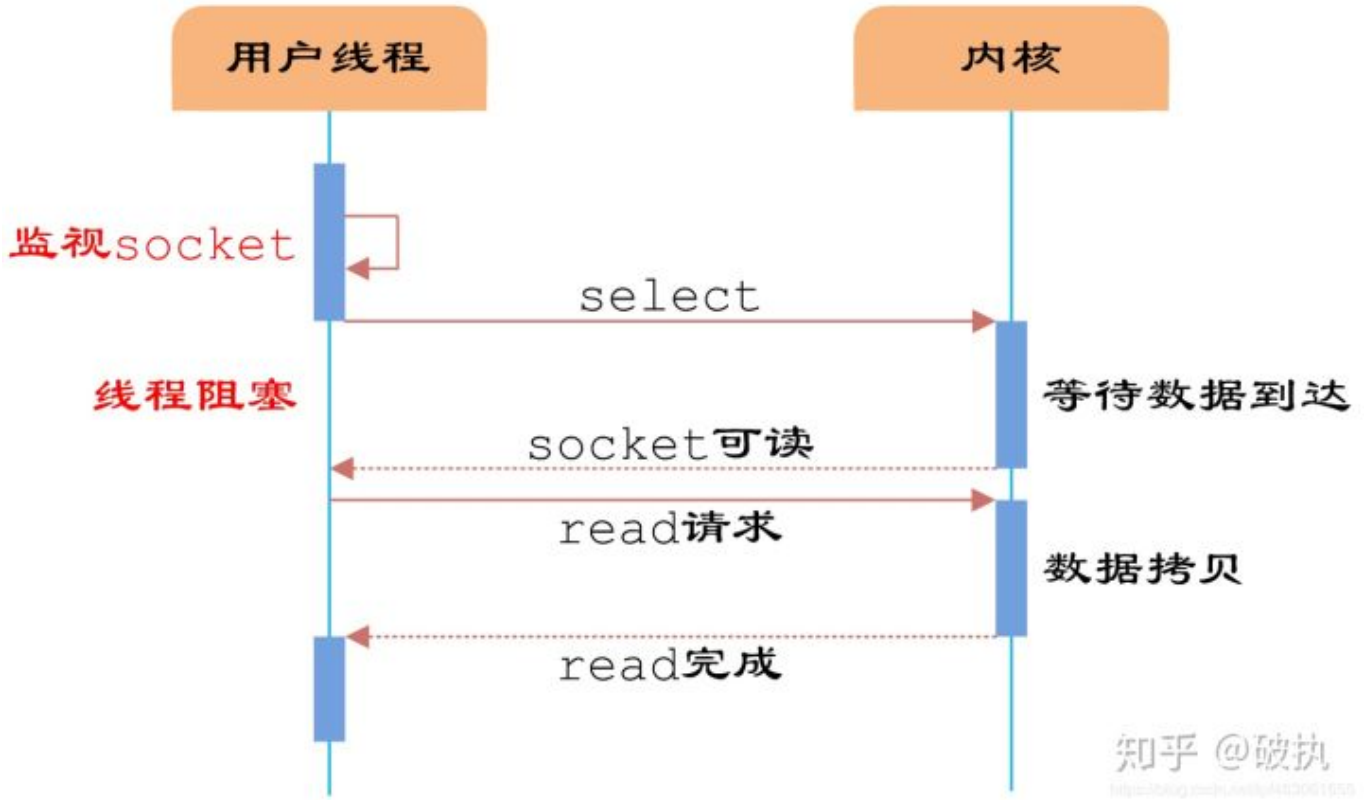
由于socket是非阻塞的方式，因此用户线程发起IO请求时立即返回。但并未读取到任何数据，用户线程需要不断地发起IO请求，直到数据到达后，才真正读取到数据，继续执行。

整个IO请求的过程中，虽然用户线程每次发起IO请求后可以立即返回，但是为了等到数据，仍需要不断地轮询、重复请求，消耗了大量的CPU的资源。**一般很少直接使用这种模型，而是在其他IO模型中使用非阻塞IO这一特性。**

IO多路复用（IO multiplexing）

建立在内核提供的多路分离函数select基础之上的，使用select函数可以避免同步非阻塞IO模型中轮询等待的问题。





首先用户线程会调用select函数来阻塞的等待数据，此处的select函数可以一次性的等待多个socket。

一旦有socket获取到数据了，该socket就会被激活，并被select函数返回。此时用户线程就可以使用这个socket来获取数据。

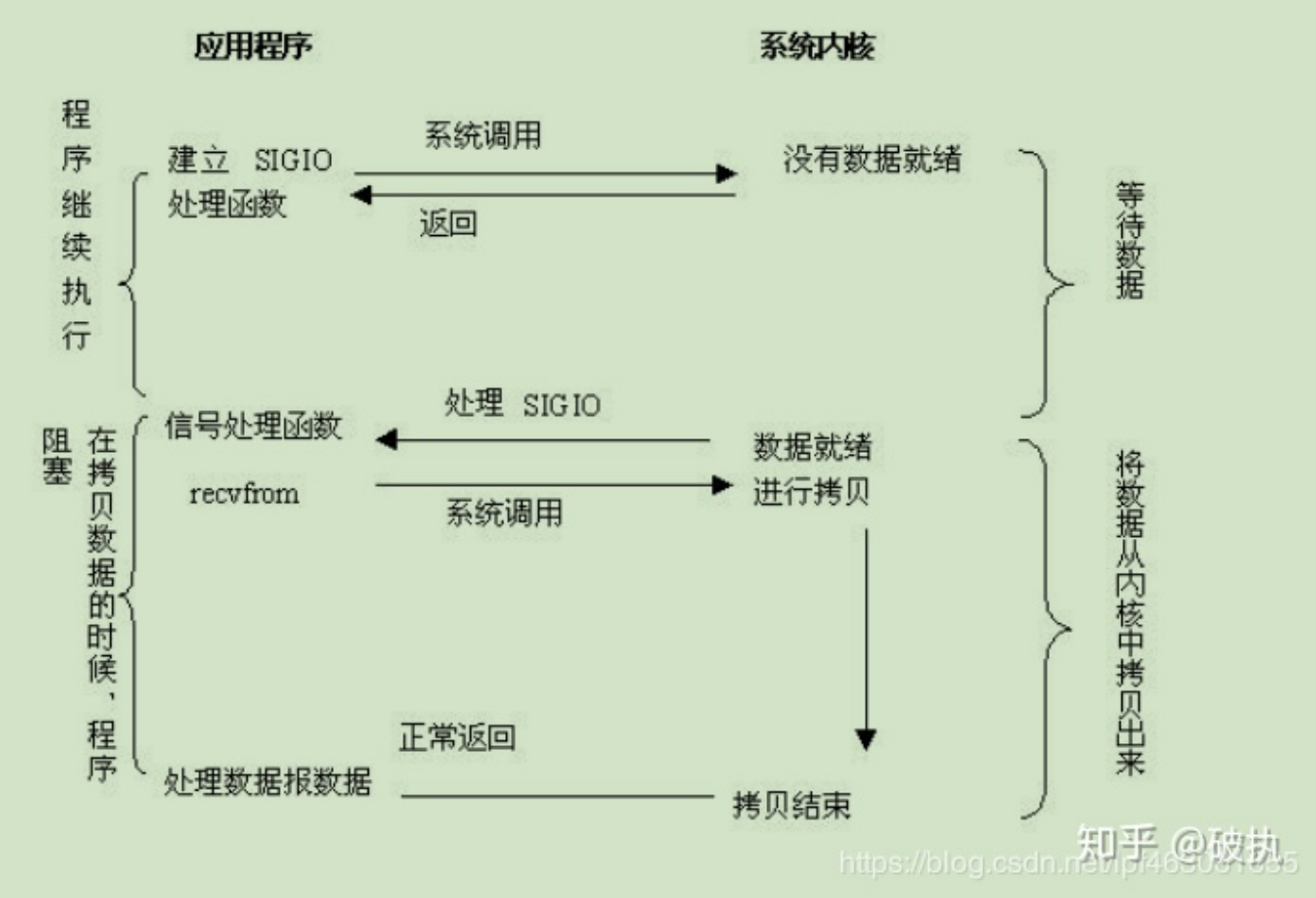
总的流程看起来跟同步阻塞模型一样。但是这种方式的 ***最大优势就是可以同时等待多个IO请求***。

用伪代码理解更容易理解：

```
{
    select(socket);
    while(1){
        sockets = select();
        for(socket in sockets) {
            if(can_read(socket)) {
                read(socket, buffer);
                process(buffer);
            }else if(can_write(socket)){
                write(socket, buffer);
                process(buffer);
            }else{
                // ....
            }
        }
    }
}
```

信号驱动IO（signal driven IO）

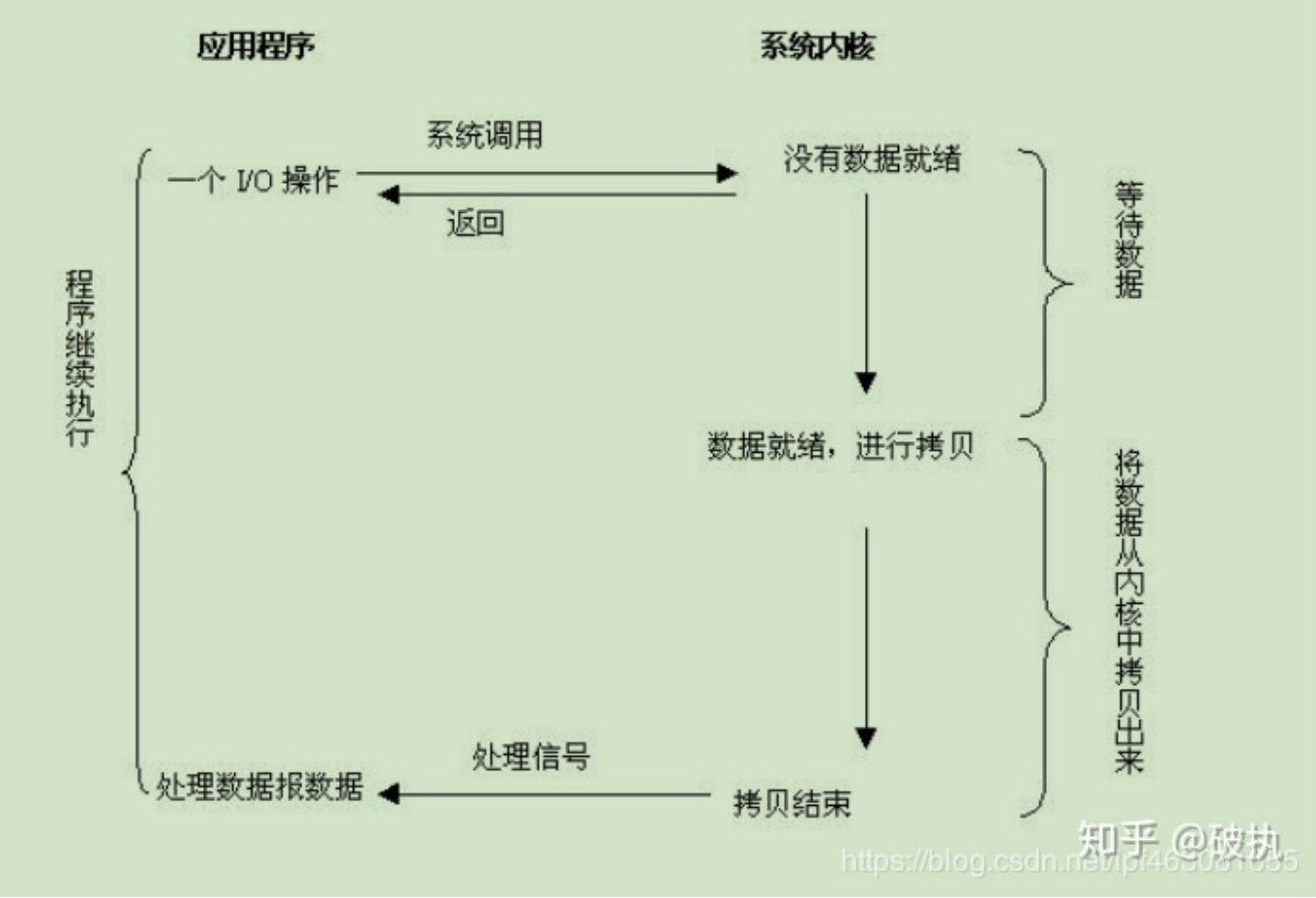
使用信号，让内核在文件描述符就绪的时候使用 SIGIO 信号来通知我们。我们将这种模式称为信号驱动 I/O 模式。



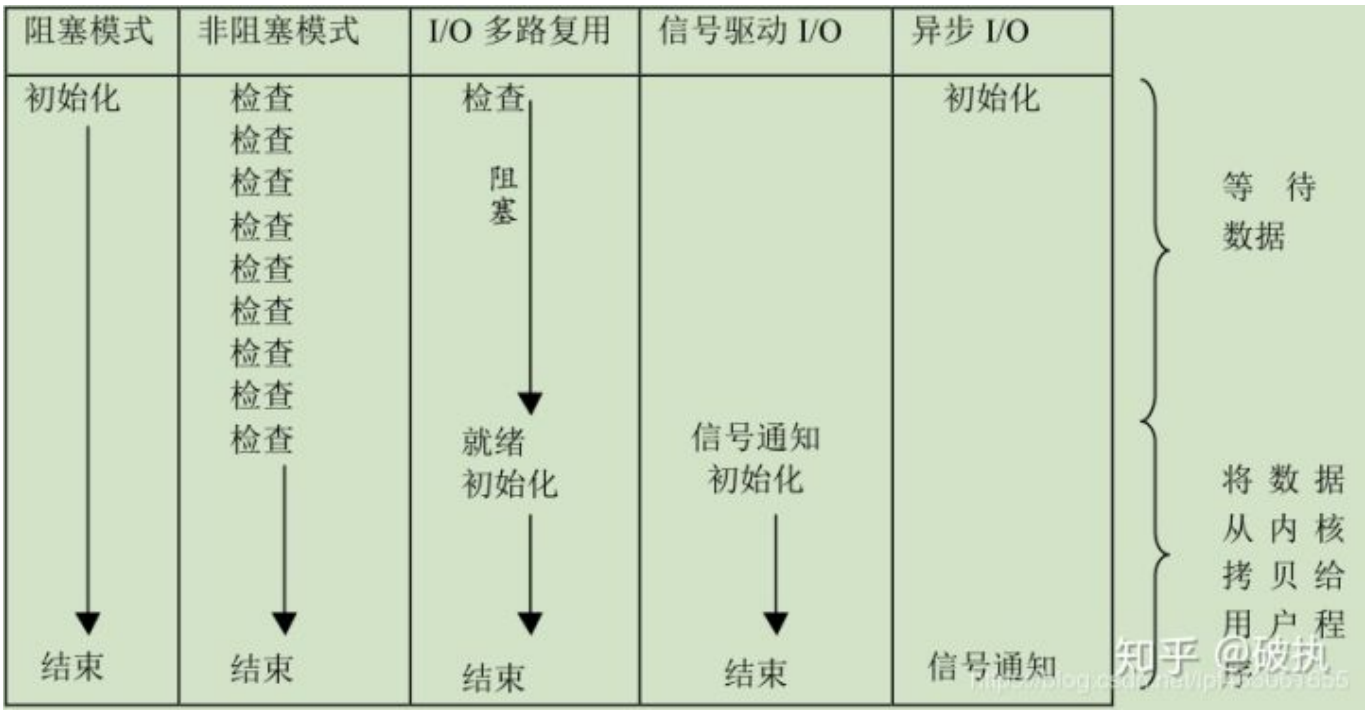
异步IO（asynchronous IO）

异步IO模式下，用户线程只需要告知内核进行IO操作即可，然后内核就会马上返回，用户线程就可以去执行其他逻辑了。

此时接受到IO请求的内核会自行完成IO的数据等待和数据拷贝，包括等待数据、拷贝数据到内核空间，拷贝数据到用户空间。之后将通知用户线程。



模型对比



五大模型，前四种都是同步的。他们的区别只是在第一阶段，第二阶段是相同的。

在数据从内核空间拷贝进用户空间的时候，线程是阻塞的（就是***前四种模型中在第二阶段的时候都是阻塞的***）。

只有在异步IO模型下，两个阶段才都是非阻塞的。

同步、异步

同步和异步描述的是用户线程和内核的交互方式。

同步：指用户线程在发起IO请求之后需要等待或者轮询内核IO操作之后才能继续执行。

异步：用户线程发起IO请求之后仍然继续执行，当内核完成IO操作之后会通知用户线程。

阻塞、非阻塞

阻塞和非阻塞描述的是用户线程调用内核IO操作的方式。

阻塞：指内核IO操作彻底完成之后才返回到用户空间。

非阻塞：指内核IO操作被调用有马上返回用户线程一个状态，无需等待IO操作彻底完成。

==同步才区分阻塞和非阻塞，异步一定是非阻塞的==