accept()：用户空间调用accept函数，进入内核空间（执行内核函数）等待连接，当TCP三 次握手建立连接后，内核将客户端的文件描述符返回给用户空间。

（用户函数->进入内核函数,线程阻塞等待连接->连接建立，返回用户函数）

read()：用户空间调用read函数，进入内核空间（执行内核函数）等待用户发送数，接收到 数据后，内核将数据返回给用户空间。

（用户函数->进入内核函数,线程阻塞等待数据->接收到数据，返回用户函数）

多路复用：用一个线程管理多个socket

1. 版本一BIO：

阻塞

线程数：单线程，既要接收连接，又要读取数据

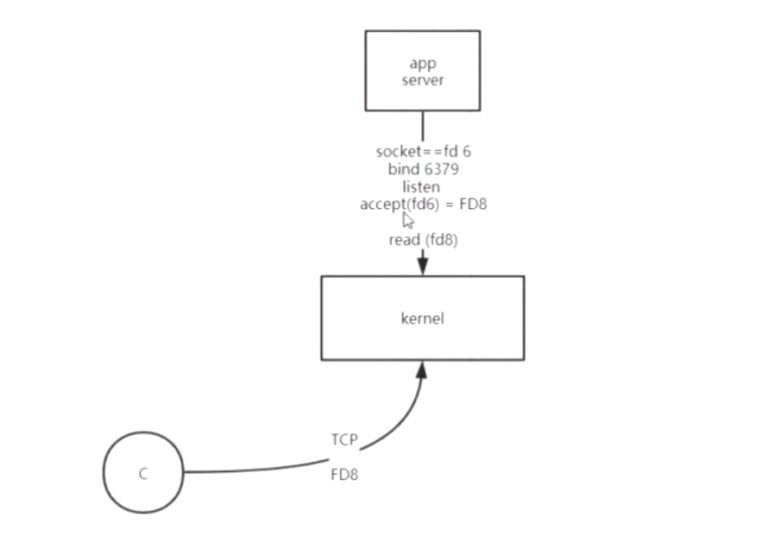
阻塞：连接（accept）阻塞，读数据（read）阻塞

{

accept();//线程阻塞

read();//线程阻塞

}



缺点：一个线程既要监听客户端，又要接收数据，且都是阻塞函数，无法支持并发。

1. 版本二BIO：解决版本一无法并发问题

阻塞

线程：多线程，主线程接收连接，子线程接收数据（一线程一连接）

阻塞：主线程accept阻塞，但read由子线程负责不阻塞

主线程只会接收连接的时候阻塞，读取数据由子线程负责

{

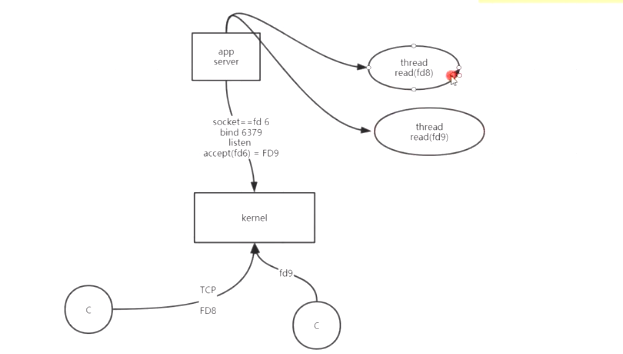
accept();//主线程阻塞

new Thread(){//主线程非阻塞

read();

}

}



优点：主线程负责接收连接，子线程负责接收数据，可支持高并发

缺点：

1. 创建线程成本高，涉及到用户态内核态切换
2. 线程栈会占用内存资源，java线程栈大小为1M，一线程一连接太费内存
3. 版本三NIO：解决版本二多线程问题，即最根本的问题：accept、read阻塞问题

轮询在用户空间

线程：单线程，既要接收连接，又要读入数据

阻塞：accept阻塞，read为非阻塞，

并且会将socket保存下来，循环遍历接收数据

{

accept();//阻塞

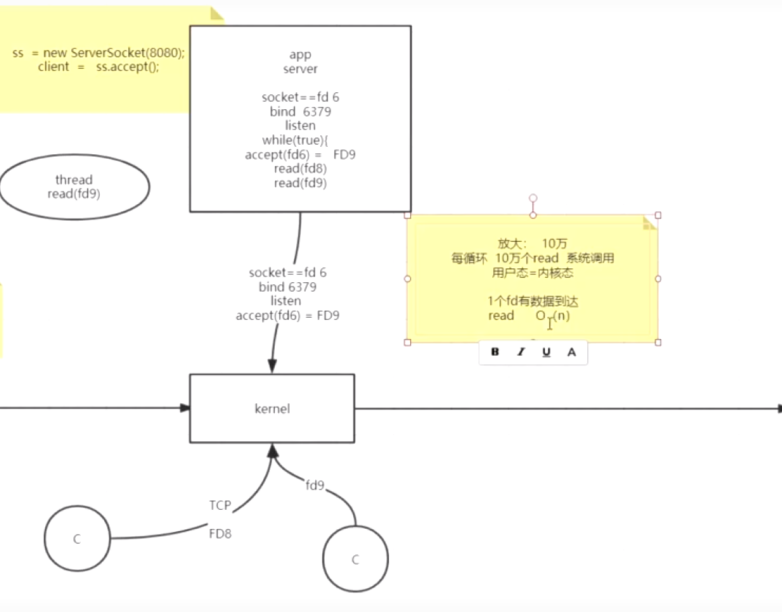
for(Socket socket:lists)

{

read(socket);//非阻塞

}

}



优点：解决多线程问题

缺点：如果有10w个socket连接

1. 循环遍历read，就会有10w次用户态和内核态切换，成本太高
2. 循环遍历socket，可能有效的就10个，空循环多，时间复杂度为o(n)
3. 版本四：解决版本三用户态与内核态频繁切换问题

轮询在内核空间

线程：单线程，既要接收连接，又要读入数据

阻塞：accept阻塞，read非阻塞

让内核循环遍历socket，是否有数据

{

accept();//阻塞

sockets = select(socketList);//让内核循环遍历socket，read；o(n)

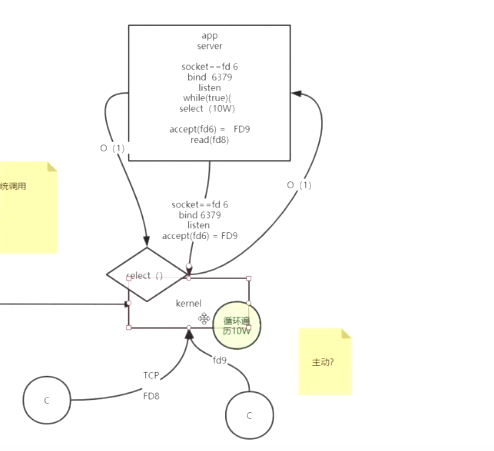
for(Socket socket:sockets )//内核返回的sockets，就是接收到数据的socket，o(1)

{

read(socket);

}

}



优点：用户态与内核态只经过一次切换，通过select函数

缺点：内核循环遍历，时间复杂度还是o(n)，空循环问题还未解决，浪费CPU

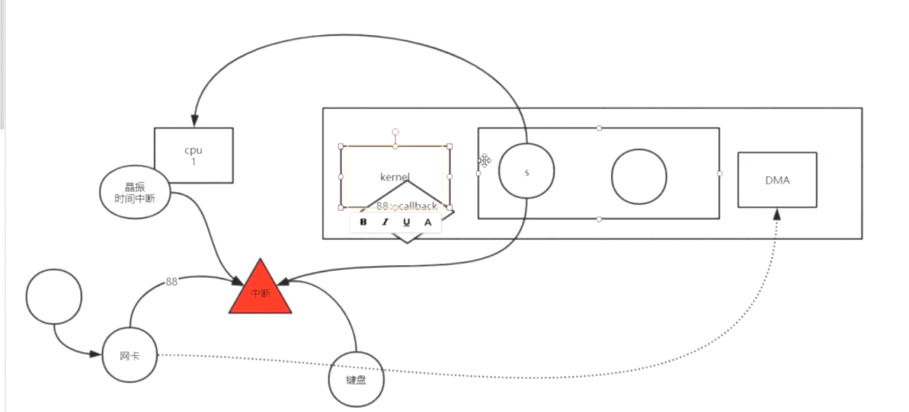
1. 版本五：解决版本四的空循环问题，epoll事件通知read、accept

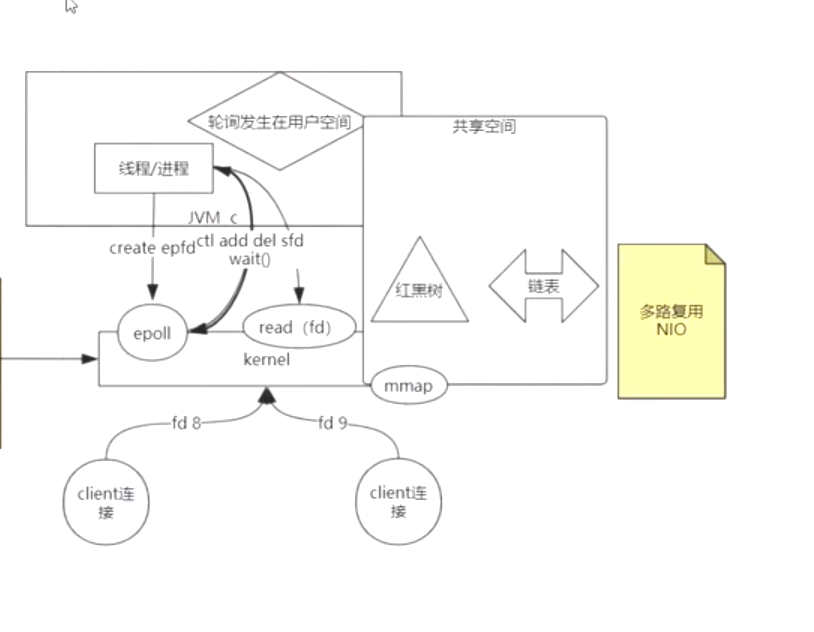
事件通知

每个硬件（网卡、键盘）都会有一个中断号，在内核可以通过中断号找到对应的函数。

中断->调用callback函数，内核就知道有事件发生

例子：网卡收到数据，直接发到DMA，然后DMA发送一个中断信号，CPU通过中断号，调用到内核里面中断号对应的回调函数。





线程：单线程，既要接收连接，又要读入数据

阻塞：accept非阻塞，read非阻塞

通过事件触发accept与read

mmap：共享空间，将sockets存储在共享空间，这样用户程序、内核都可以看到，减少了把数据socketlist从用户程序拷贝到内核空间。

共享空间的红黑树：存储文件描述符（socket ID）

共享空间的链表：文件描述符与事件（read、accept）

观察者模式：注册----->等待通知----->处理

Epoll：

创建Epoll：epoll\_create()

注册事件：epoll\_ctl()

等待事件：epoll\_wait()

//fd为文件描述符，即socket id

{

fd5 = Epoll\_create();

Socket fd\_server = new Socket();

epoll\_create(add,fd\_server,accept);//注册接收连接事件

while(Epoll\_wait(event))//根据事件触发accept、read

{

switch(event,fd)

{

case accpt\_event:

fd\_listen = accept();//肯定有连接，相当于非阻塞

epoll\_create(add,fd\_listen,read);//注册接收数据事件

case read\_event:

read();//肯定有数据，相当于非阻塞

}

}

accept();//阻塞

sockets = select(socketList);//让内核循环遍历socket，read；o(n)

for(Socket socket:sockets )//内核返回的sockets，就是接收到数据的socket，o(1)

{

read(socket);

}

}

优点：解决空循环问题

