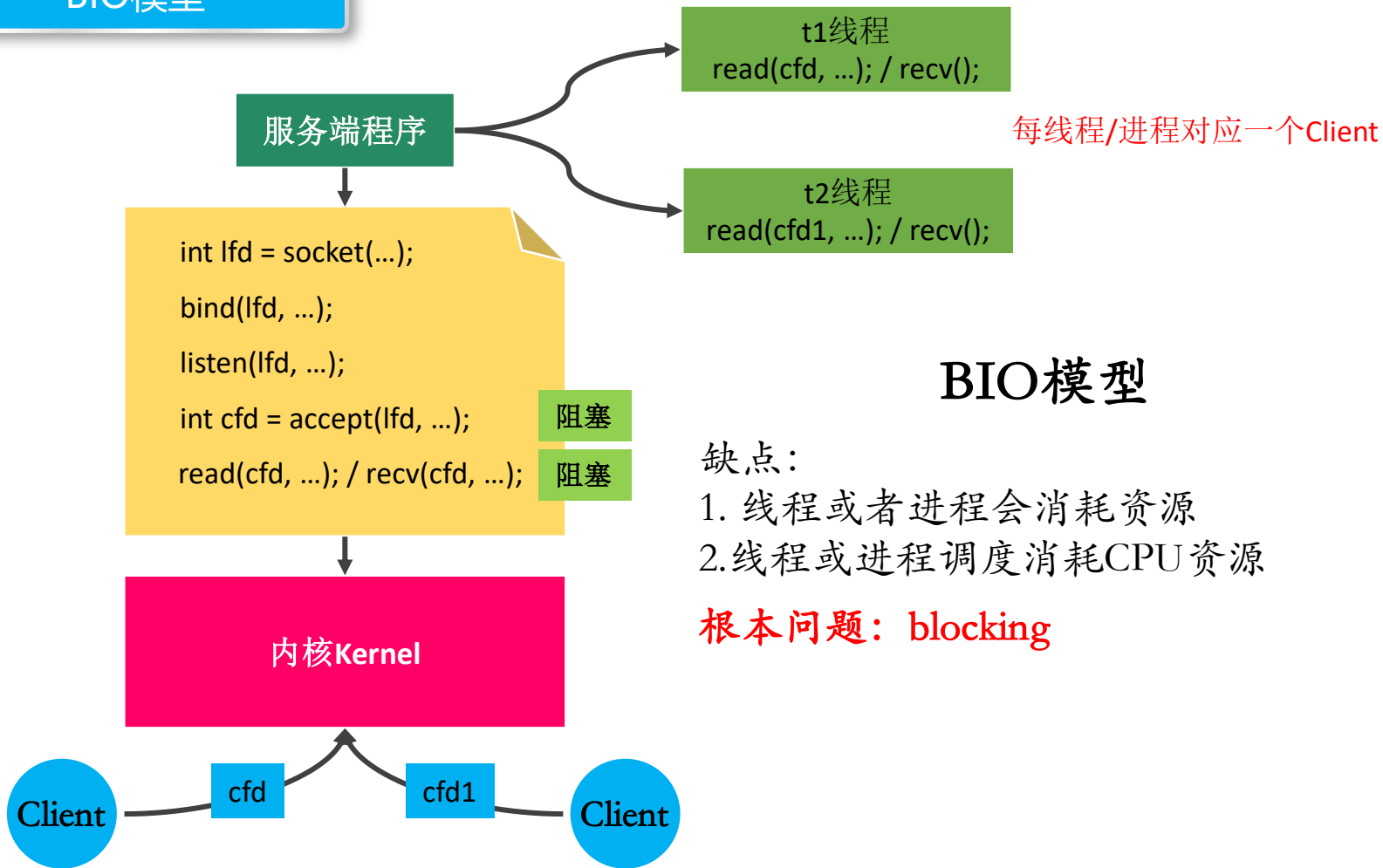


好处：不占用CPU宝贵的时间片  
缺点：同一时刻只能处理一个操作, 效率低

## 多线程或者多进程解决

缺点：  
1. 线程或者进程会消耗资源  
2. 线程或进程调度消耗CPU资源

# BIO模型



## BIO模型

缺点:

1. 线程或者进程会消耗资源
2. 线程或进程调度消耗CPU资源

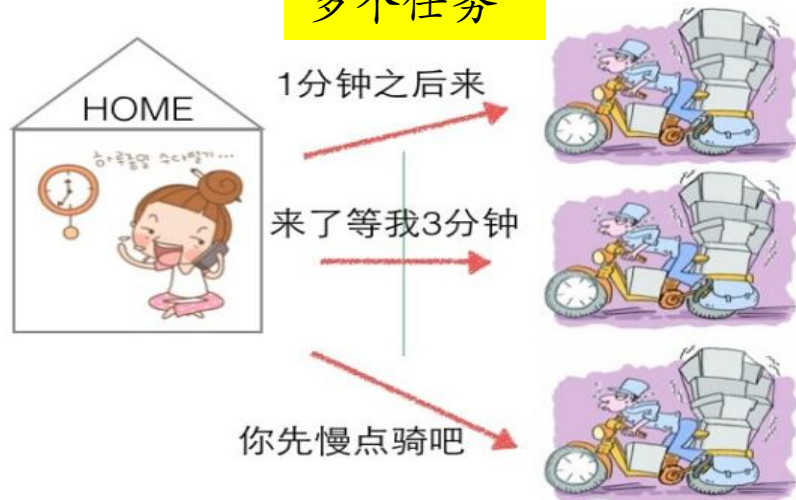
根本问题: blocking

## 非阻塞, 忙轮询

### 一个任务



### 多个任务

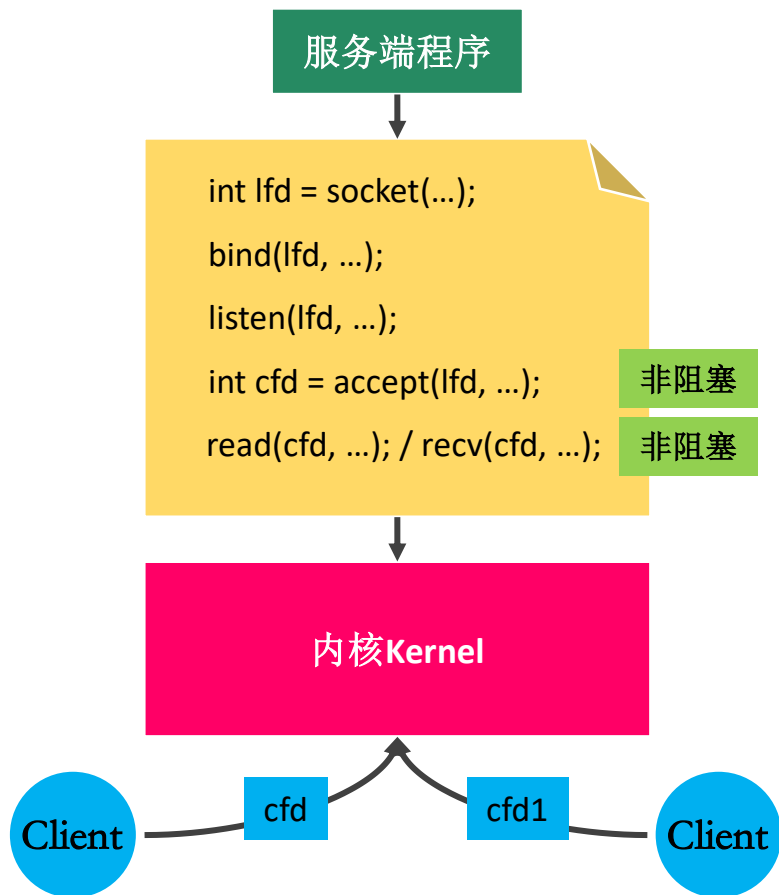


优点: 提高了程序的执行效率

缺点: 需要占用更多的CPU和系统资源

使用IO多路转接技术select/poll/epoll

## NIO模型



## NIO模型

1W Client

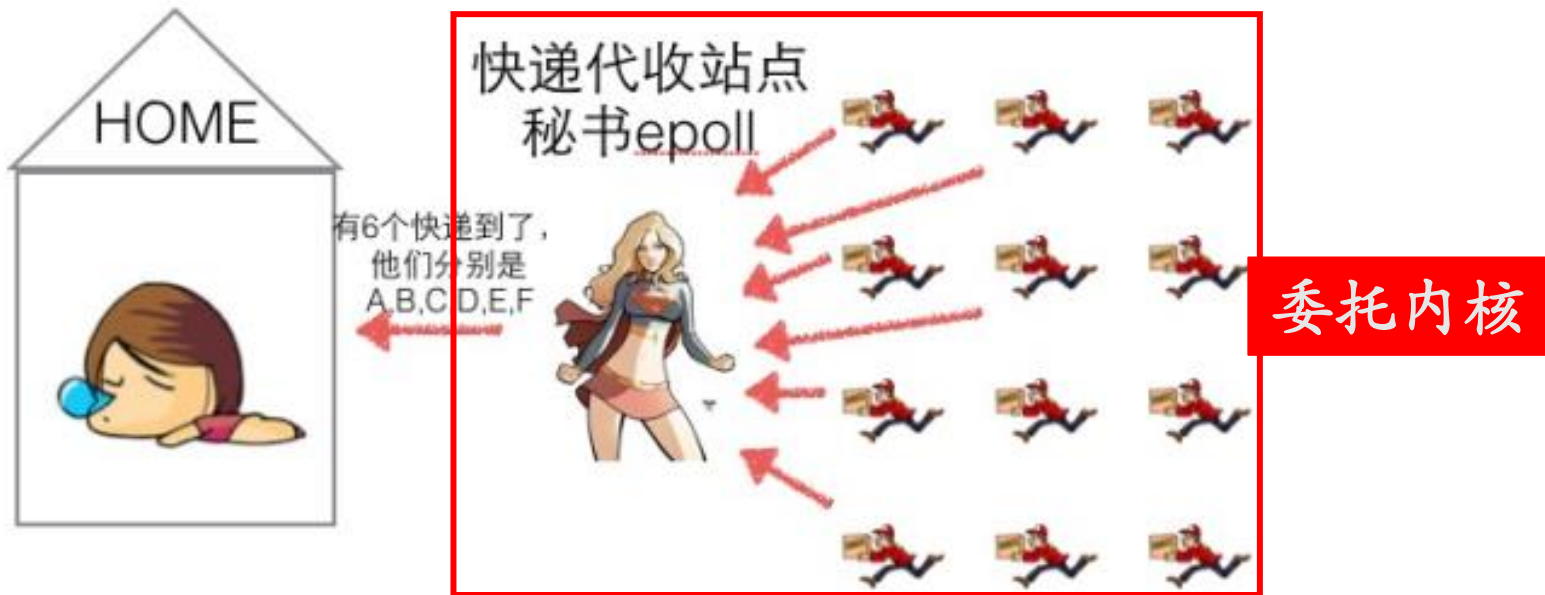
每循环内  $O(n)$  系统调用

第一种：select/poll



select代收员比较懒，她只会告诉你有几个快递到了，但是哪个快递，你需要挨个遍历一遍。

### 第二种：epoll

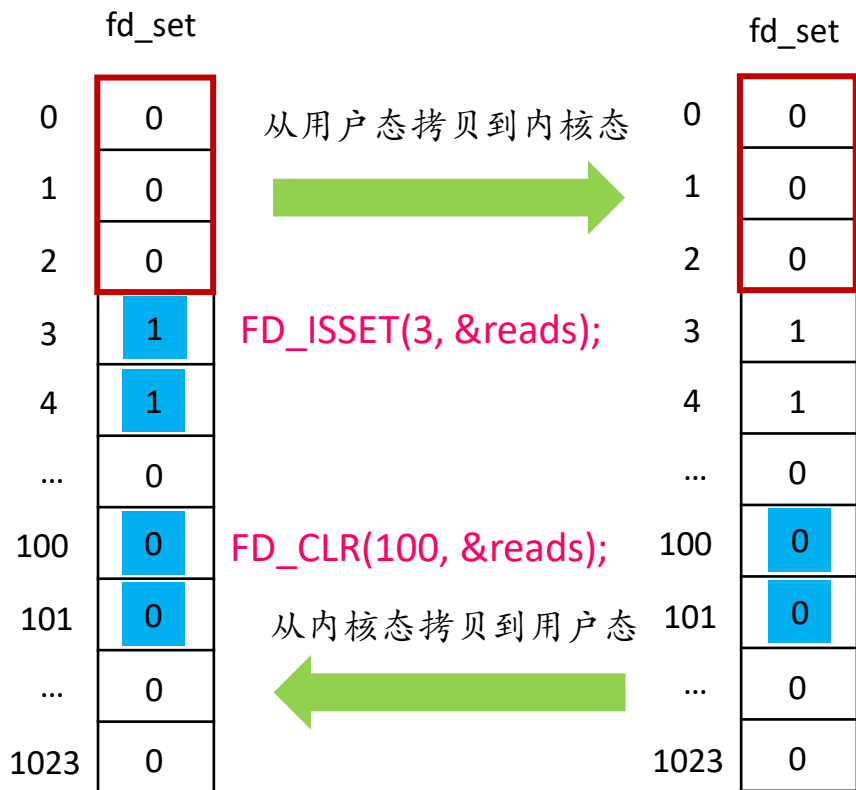


epoll代收快递员很勤快，她不仅会告诉你有几个快递到了，还会告诉你是哪个快递公司的快递

## select()工作过程分析

客户端A,B,C,D连接到服务器分别对应文件描述符3,4,100,101

A,B发送了数据



```
fd_set reads;
```

```
FD_SET(3, &reads);
```

```
FD_SET(4, &reads);
```

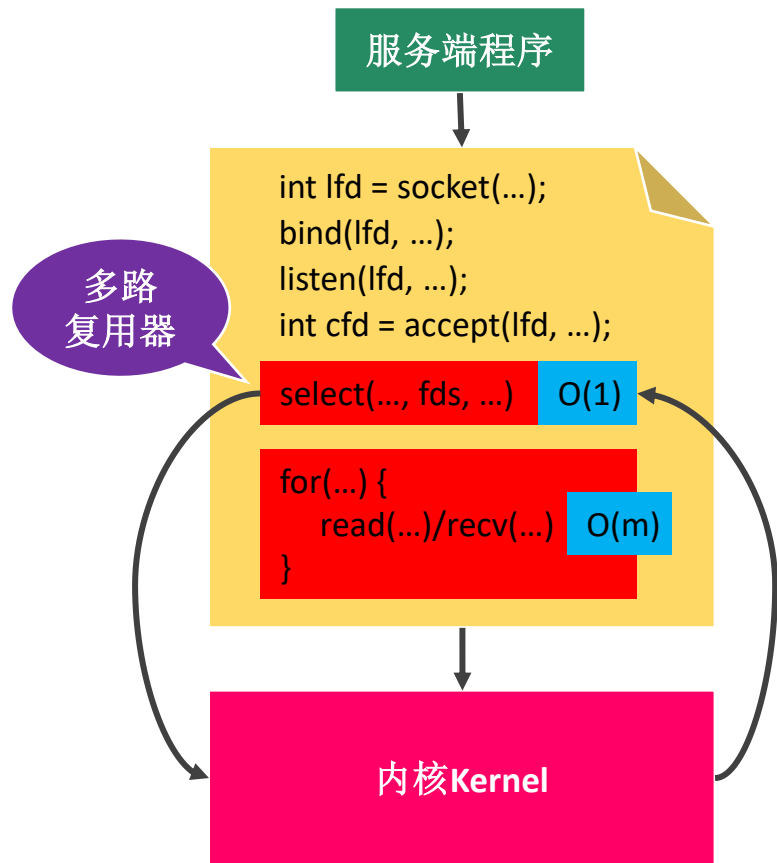
```
FD_SET(100, &reads);
```

```
FD_SET(101, &reads);
```

```
select(101+1, &reads, NULL, NULL, NULL);
```

```
for(...) {  
    read(...)/recv(...)  
}
```

## select()多路复用

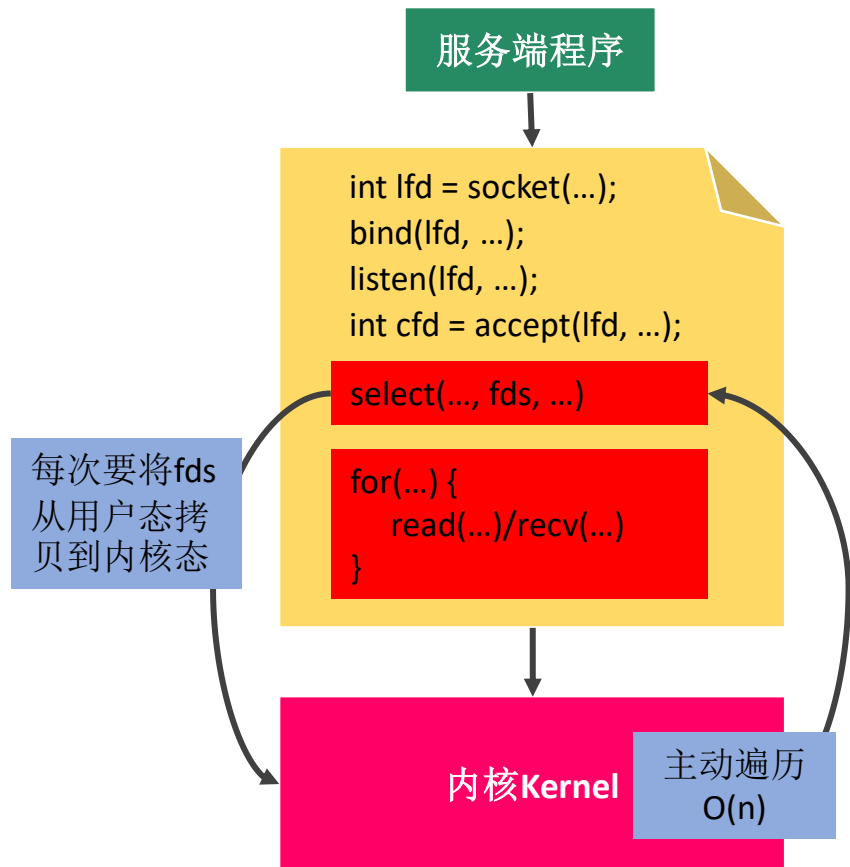


```
int select(int nfds,  
           fd_set *readfds,  
           fd_set *writefds,  
           fd_set *exceptfds,  
           struct timeval *timeout);
```

```
FD_SET(int fd, fd_set* set);  
FD_CLR(int fd, fd_set* set);  
FD_ISSET(int fd, fd_set *set);  
FD_ZERO(fd_set *set);
```



## select()的缺点



缺点:

1. 每次调用select, 都需要把fd集合从用户态拷贝到内核态, 这个开销在fd很多时会很大

2. 同时每次调用select都需要在内核遍历传递进来的所有fd, 这个开销在fd很多时也很大

3. select支持的文件描述符数量太小了, 默认是1024

4. fds集合不能重用, 每次都需要重置, 原因是每次会cp回内核, 可能会导致一些连接到的状态置为0, 不会在检测这些。

## poll()多路复用

```
int poll(struct pollfd *fd, nfds_t nfds, int timeout);  
int select(  
    int nfds,  
    fd_set *readfds,  
    fd_set *writefds,  
    fd_set *exceptfds,  
    struct timeval *timeout);
```

```
struct pollfd {  
    int fd;  
    short events;  
    short revents;  
};
```

事件	常值	作为events的值	作为revents的值	说明
读事件	POLLIN	✓	✓	普通或优先带数据可读
	POLLRDNORM	✓	✓	普通数据可读
	POLLRDBAND	✓	✓	优先级带数据可读
	POLLPRI	✓	✓	高优先级数据可读
写事件	POLLOUT	✓	✓	普通或优先带数据可写
	POLLWRNORM	✓	✓	普通数据可写
	POLLWRBAND	✓	✓	优先级带数据可写
错误事件	POLLERR		✓	发生错误
	POLLHUP		✓	发生挂起
	POLLNVAL		✓	描述不是打开的文件

## epoll()多路复用

```
struct epoll_event ev;  
ev.events = EPOLLIN;  
ev.data.fd = lfd;
```

服务端程序

```
int lfd = socket(...);  
bind(lfd, ...);  
listen(lfd, ...);  
int cfd = accept(lfd, ...);  
int epfd = epoll_create(2000);  
epoll_ctl(epfd, EPOLL_CTL_ADD, lfd, &ev);  
.....  
int ret = epoll_wait(epfd, ...);  
  
for(...) {  
    read(...)/recv(...)  
}
```

```
struct eventpoll{  
    ....  
    struct rb_root rbr;  
    struct list_head rdlist;  
    ....  
};
```

内核中的数据结构

eventpoll

rbr

rdlist

红黑树

双链表

