# IO函数

## I/O读写函数

### recv

使用recv函数接收数据：

#include <sys/socket.h>

ssize\_t recv(int fd,void\* buf,size\_t len,int flags);

其中，buf是用户接收缓冲区的地址，len表示接收数据长度，recv函数从内核的接收缓冲区中复制数据到用户指定的缓冲区中：当内核中的数据比指定的缓冲区小时，一般会复制内核缓冲区的所有数据到用户缓冲区，并返回数据的长度；当内核接收缓冲区的数据比用户指定的长度多时，会将用户指定len长度的数据复制到用户缓冲区，其余数据等下次调用的时候复制。内核在复制完内核缓冲区的数据后，会销毁该数据。

其中flag的取值：

 recv函数返回错误时，错误码的含义：

recv函数的特点：

1. recv()常用于TCP连接，recvfrom()用于UDP连接（在数据报套接字DUP绑定了地址和端口后，也可以使用recv接收数据）；
2. 可以使用flag参数使其适用于多种场景；
3. 需要根据返回值来判断数据是否接收完毕。

### send

使用send函数发送数据：

#include <sys/socket.h>

ssize\_t send(int fd,const void\* buf,size\_t len,int flags);

send()函数的特点：

需要根据返回值来判断数据是否发送完毕

send()用于处理连接状态的描述符

flag标志位使得其行为多样

send函数errno含义：



### readv

#include <sys/uio.h>

ssize\_t readv(int fd,struct iovec\* vector,int count);

ssize\_t writev(int fd,const struct iovec\* vector,int count);

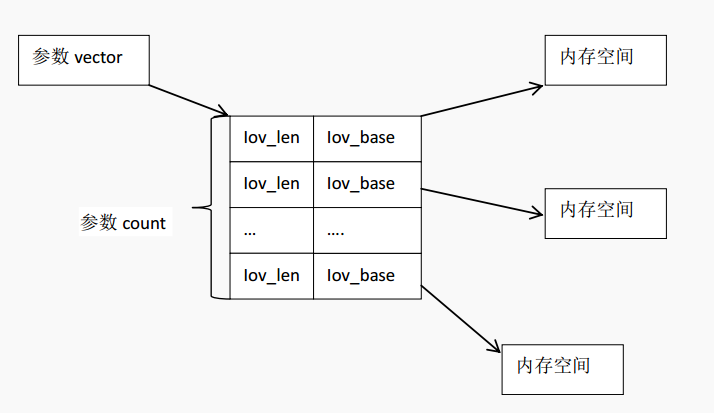
这两个函数的第二个参数都是一个指向某个iovec结构数组的一个指针：

struct iovec{

void\* iov\_base; //缓冲开始地址

size\_t iov\_len; //缓冲长度

};



### writev

### recvmsg

#include <sys/socket.h>

ssize\_t recvmsg(int fd,struct msghdr\* msg,int flags);

ssize\_t sendmsg(int fd,struct msghdr\* msg,int flags);

### sendmsg

### I/O函数对比



## I/O操作函数

### pipe

创建管道实现进程间通信函数pipe()：

int pipe(int fd[2]);

使用pipe()函数需要注意的事项：

fd[0]用于读取数据，fd[1]用于写数据

fd[0]和fd[1]默认情况下为阻塞

关闭管道的依据是引用计数

写失败触发SIGPIPE信号

pipe()函数创建管道的特性：

管道内部传输的数据流是字节流

单向管道

管道容量有限制

基于上述缺陷，内核提供了sockerpair创建双向的管道

### socketpair

创建双向管道函数socketpair()：

int socketpair(int domain,int type,int protocol,int fd[2]);

### mmap

创建共享内存函数mmap()：

void\* mmap(void\* start,size\_t length,int prot,int flags,int fd,off\_t

offset);

int munmap(void\* start,size\_t length);

mmap()函数port参数用来设置内存段的访问权限：

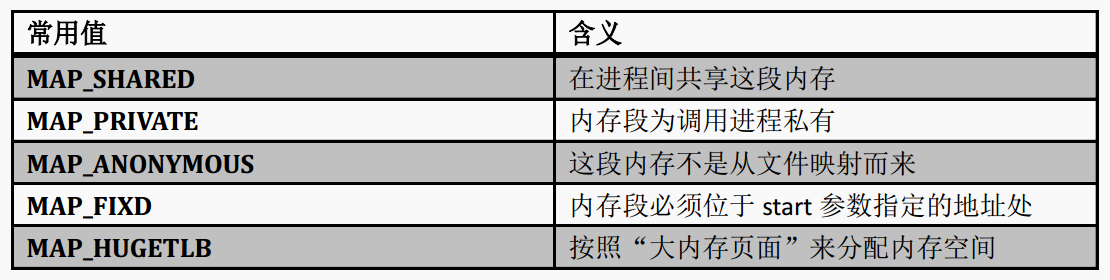
PROT\_READ：内存段可读

PORT\_WRITE：内存段可写

PORT\_EXEC：内存段可执行

PORT\_NONE：内存段不能被访问

参数flag的取值：



### fcntl

用于修改描述符属性的函数fcntl()：

int fcntl(int fd,int cmd,…);

使用fcntl函数获取和设置文件描述符的状态标志：

F\_GETFL：获取fd的状态标志

F\_SETFL：设置fd的状态标志

注：ioctl比fcntl有更多控制行为，但是对于文件描述符的控制，使用fncntl。

### splice

用于两个文件描述符之间移动数据的函数splice()：

ssize\_t splice(int fd\_in,loff\_t\* off\_in,int fd\_out,loff\_t\*

off\_out,size\_t len,unsigned int flags);

flag参数的含义：



使用splice()函数的注意事项：

fd\_in和fd\_out必须至少有一个是管道文件描述符

返回值为0时，表示没有数据需要移动

### 实例

#include <sys/socket.h>

#include <sys/types.h>

#include <stdlib.h>

#include <stdio.h>

int main()

{

int fd[2];

int ret = socketpair(AF\_UNIX,SOCK\_STREAM,0,fd);

if(ret < 0)

{

perror("socketpair() error\r\n");

exit(1);

}

if(fork())

{

printf("parent process \r\n");

int val = 0;

close(fd[1]);

while(1)

{

sleep(1);

++val;

printf("send val to child : %d\r\n",val);

write(fd[0],&val,sizeof(val));

read(fd[0],&val,sizeof(val));

printf("read val from child : %d \r\n",val);

}

}

else

{

printf("child process \r\n");

int val;

close(fd[0]);

while(1)

{

read(fd[1],&val,sizeof(val));

++val;

write(fd[1],&val,sizeof(val));

}

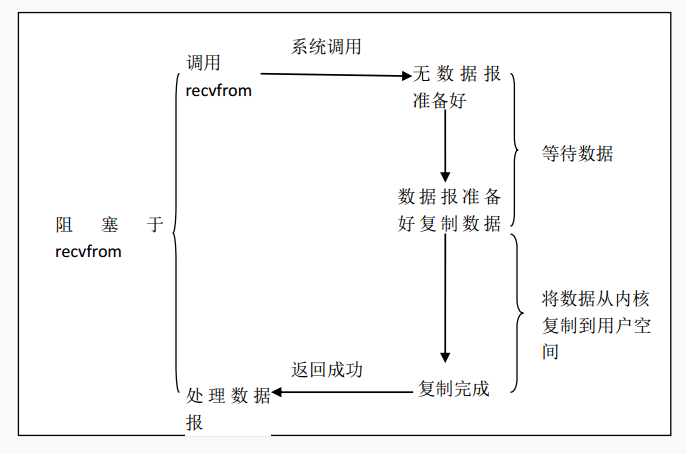
}

}

# IO模型

## 阻塞式I/O

### 原理



### 分类

可能阻塞套接字的系统调用分为以下几类：

输入操作中的read()、readv()、recv()、recvfrom()

输出操作中的write()、writev()、send()、sendto()

接收连接中的accept()

发起连接中的connect()

### 特点

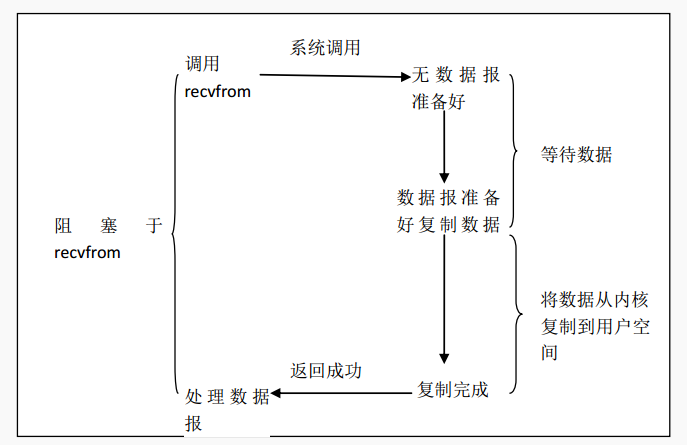
阻塞式I/O的缺点：

效率低

不适合高并发的开发

## 非阻塞式I/O

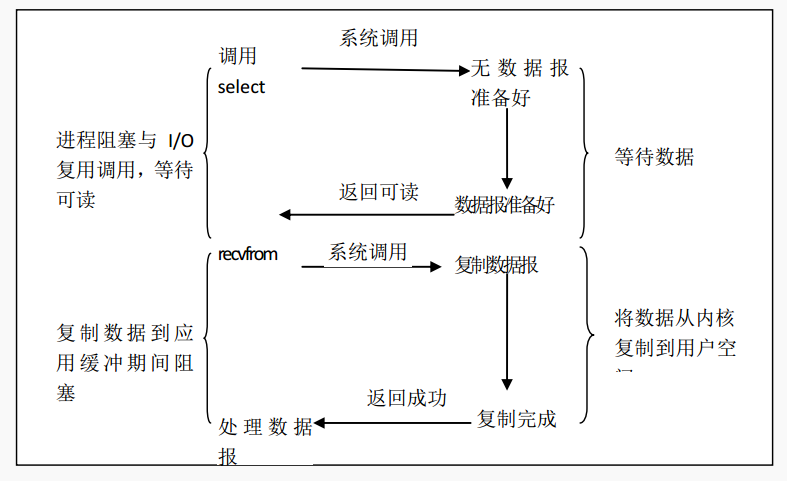
### 原理



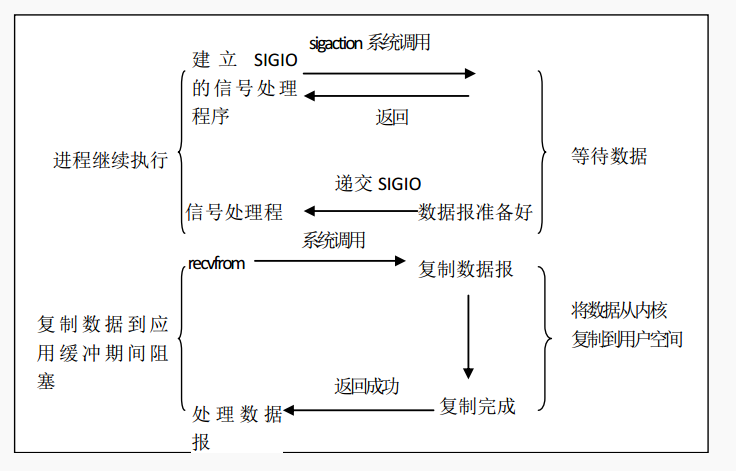
### 非阻塞accept

### 非阻塞connect

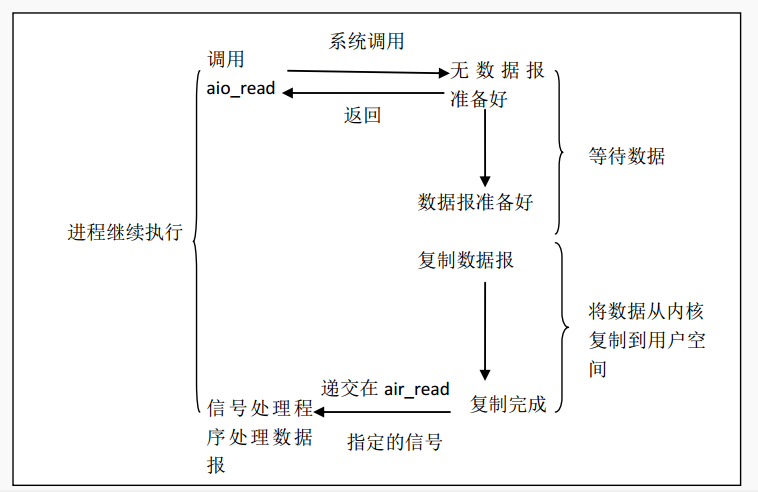
## I/O复用



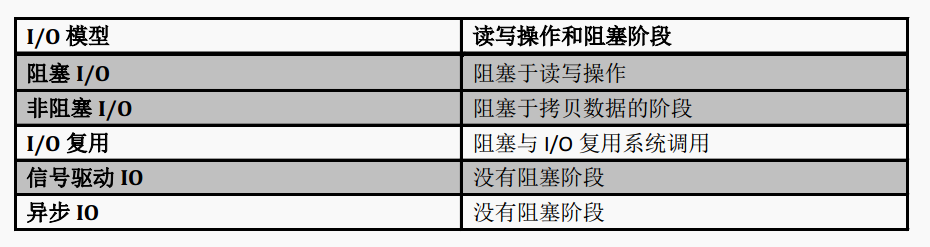
## 信号驱动式I/O



## 异步I/O



## I/O模型比较



### 同步/异步

同步和异步与消息的通知机制有关：

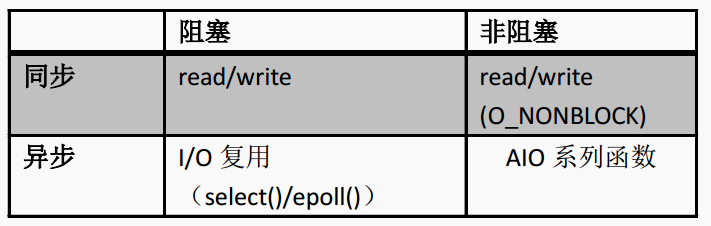
同步：在发出一个功能调用时，在没有得到结果之前，该调用就不返回

异步：异步过程调用发出后，调用者不能立刻得到结果

同步和异步的使用特点：

关注的是消息如何通知的机制

异步机制中，消息通知和消息处理之间需要一个桥梁



### 阻塞/非阻塞

阻塞和非阻塞和等待消息时的状态有关：

阻塞：函数之余在得到结果之后才会返回

非阻塞：不能立刻得到结果的情况下，调用会立刻返回