**实验 五种I/O模型和select实现回射服务器端**

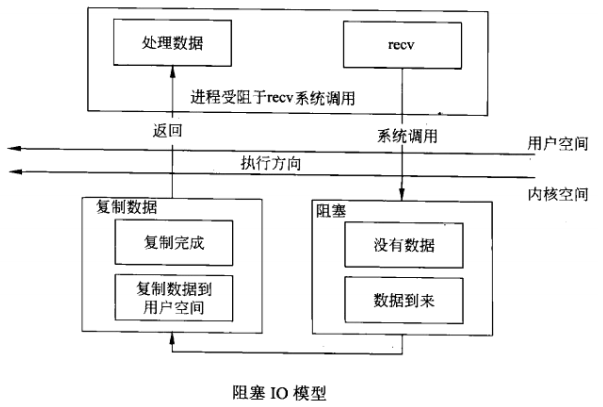
**实验目的： 用select 改写回射服务器端**

**实验步骤**

**https://blog.csdn.net/jnu\_simba/article/details/9071445**

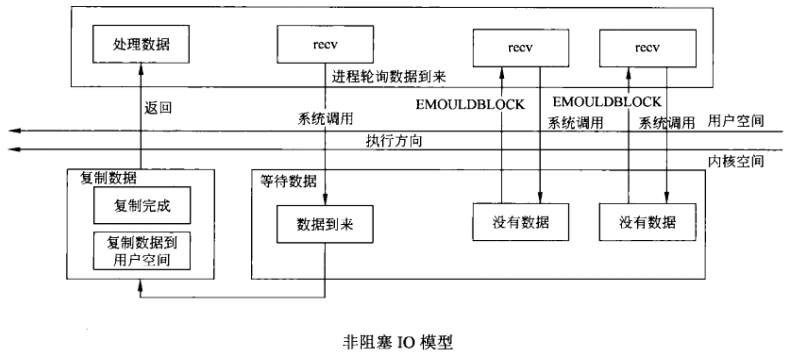
**一、五种I/O模型**

**1、阻塞I/O**

****

**我们在前面所说的I/O模型都是阻塞I/O，即调用recv系统调用，如果没有数据则阻塞等待，当数据到来则将数据从内核空间（套接口缓冲区）拷贝到用户空间（recv函数提供的buf），然后recv返回，进行数据处理。**

**非阻塞I/O**

****

**我们可以使用 fcntl(fd, F\_SETFL, flag | O\_NONBLOCK); 将套接字标志变成非阻塞，调用recv，如果设备暂时没有数据可读就返回-1，同时置errno为EWOULDBLOCK（或者EAGAIN，这两个宏定义的值相同），表示本来应该阻塞在这里（would block，虚拟语气），事实上并没有阻塞而是直接返回错误，调用者应该试着再读一次（again）。这种行为方式称为轮询（Poll），调用者只是查询一下，而不是阻塞在这里死等，这样可以同时监视多个设备：**

**while(1)**

**{**

**非阻塞read(设备1);**

**if(设备1有数据到达)**

**处理数据;**

**非阻塞read(设备2);**

**if(设备2有数据到达)**

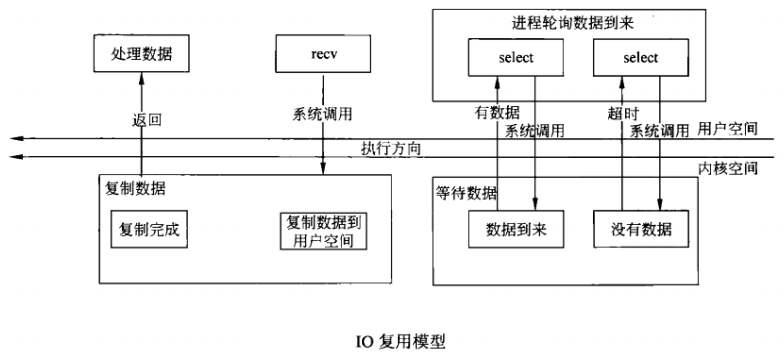
**处理数据;**

**}**

**如果read(设备1)是阻塞的，那么只要设备1没有数据到达就会一直阻塞在设备1的read调用上，即使设备2有数据到达也不能处理，使用非阻塞I/O就可以避免设备2得不到及时处理。**

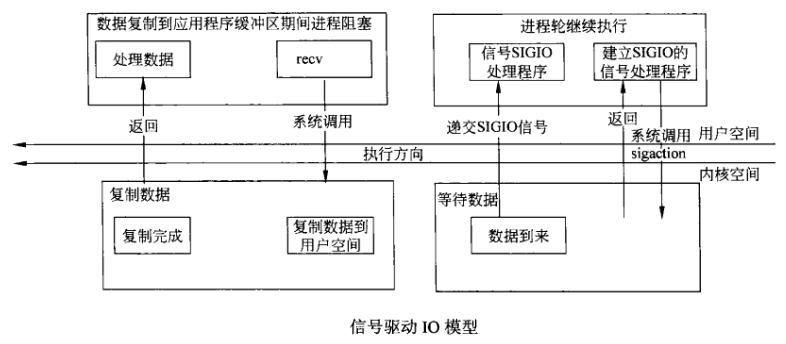
**非阻塞I/O有一个缺点，如果所有设备都一直没有数据到达，调用者需要反复查询做无用功，如果阻塞在那里，操作系统可以调度别的进程执行，就不会做无用功了，在实际应用中非阻塞I/O模型经常与IO multiplexing 一起使用。**

**3、I/O复用方式：select**

****

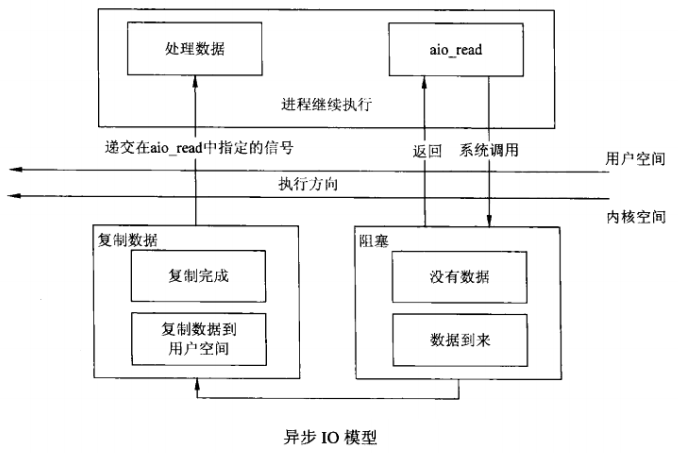
**用select来管理多个I/O，当没有数据时select阻塞，如果在超时时间内数据到来则select返回，再调用recv进行数据的复制，recv返回后处理数据。**

**4、信号驱动I/O**

****

**先注册SIGIO信号的处理函数，进程继续执行其他操作，当数据到来时会发送SIGIO信号给进程，然后可以在信号处理函数中调用recv进行数据的复制，然后recv返回进行数据处理。**

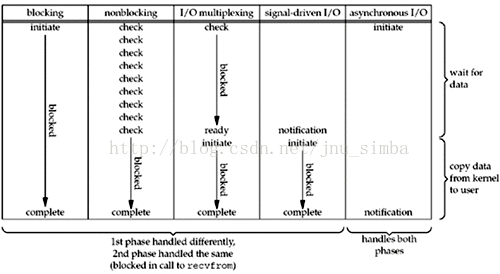
**5、异步I/O**

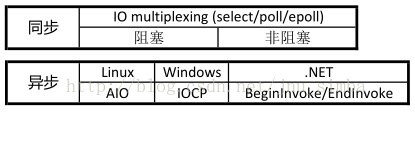
****

**aio\_read 函数也会提供一个buf，系统调用进入内核，如果没有数据则立即返回，进程继续执行其他操作，所以叫异步I/O，当数据到来时内核自动复制数据，然后推送给用户空间，通过在aio\_read中指定的信号通知进程，让其处理数据。异步I/O跟信号驱动I/O的不同之处在于，它不用调用recv进行数据的复制，如果将后者比做”拉pull“，则前者可以认为是”push推“，push的效率会高点，其实异步I/O跟windows下面的完成端口差不多，但aio\_read的实现或多或少存在问题，用得也比较少。实践中用得比较多的如boost 库的asio 也是异步IO。**

**脚注：同步和异步的区别在于是不是要求处理消息者自己来完成将数据从内核缓冲区复制回进程缓冲区的过程。消息者阻塞和非阻塞应该是发生在消息的处理的时刻。阻塞其实就是等待，发出通知，等待结果完成。非阻塞属于发出通知，立即返回结果，没有等待过程。**

**对unix来讲：阻塞式I/O(默认)，非阻塞式I/O(nonblock)，I/O复用(select/poll/epoll)，信号驱动IO都属于同步I/O，因为它们在数据由内核空间复制回进程缓冲区时都是阻塞的(不能干别的事)。只有异步I/O模型(AIO)是符合异步I/O操作的含义的，即在1数据准备完成、2由内核空间拷贝回缓冲区后 通知进程，在等待通知的这段时间里可以干别的事。**

****

****

**POSIX defines these two terms as follows:**

**A synchronous I/O operation causes the requesting process to be blocked until that I/O operation completes.**

**An asynchronous I/O operation does not cause the requesting process to be blocked.**

**Using these definitions, the first four I/O models—blocking, nonblocking, I/O multiplexing, and signal-driven I/O—are all synchronous because the actual I/O operation (recvfrom) blocks the process. Only the asynchronous I/O model matches the asynchronous I/O definition.**

**二、select函数简介**

**/\* According to POSIX.1-2001 \*/**

**#include <sys/select.h>**

**/\* According to earlier standards \*/**

**#include <sys/time.h>**

**#include <sys/types.h>**

**#include <unistd.h>**

**int select(int nfds, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, struct timeval \*timeout);**

**参数1：读写异常集合中的文件描述符的最大值加1；**

**参数2：读集合，关心可读事件；**

**套接口缓冲区有数据可读**

**对等连接的写一半关闭。即接收到FIN段，读操作将返回0**

**如果是监听套接口，已完成连接队列不为空时。**

**套接口上发生了一个错误待处理，错误可以通过getsockopt指定SO\_ERROR选项来获取。**

**参数3：写集合，关心可写事件；**

**套接口发送缓冲区有空间容纳数据。（连接一旦建立就可写）**

**对等连接的读一半关闭。即收到RST段之后，再次调用write操作。**

**套接口上发生了一个错误待处理，错误可以通过getsockopt指定SO\_ERROR选项来获取。**

**参数4：异常集合，关心异常事件；**

**套接口存在带外数据（TCP头部 URG标志，16位紧急指针字段）**

**参数5：超时时间结构体**

**对于参数2，3，4来说，如果不关心对应事件则设置为NULL即可。注意5个参数都是输入输出参数，即select返回时可能对其进行了修改，比如集合被修改以便标记哪些套接口发生了事件，时间结构体的传出参数是剩余的时间，如果设置为NULL表示永不超时。用select管理多个I/O，select阻塞等待，一旦其中的一个或多个I/O检测到我们所感兴趣的事件，select函数返回，返回值为检测到的事件个数，并且返回哪些I/O发送了事件，遍历这些事件，进而处理事件。注意当select阻塞返回后，此时调用accept 接收连接是不会阻塞的，直接返回已连接套接字，可以认为是select 提前阻塞了。但此时调用write 还是可能阻塞的，因为需要写入的空间大小可能缓冲区还不满足。**

**下面是4个可以对集合进行操作的宏：**

**void FD\_CLR(int fd, fd\_set \*set); // 清除出集合**

**int  FD\_ISSET(int fd, fd\_set \*set); // 判断是否在集合中**

**void FD\_SET(int fd, fd\_set \*set); // 添加进集合中**

**void FD\_ZERO(fd\_set \*set); // 将集合清零**

**RETURN VALUE**

**On success, select() return the number of file descriptors contained in the three returned descriptor sets (that is,**

**the total number of bits that are set in readfds, writefds, exceptfds) which may be zero if the timeout  expires  before  anything**

**interesting  happens.   On error, -1 is returned, and errno is set appropriately; the sets and timeout become undefined, so do not**

**rely on their contents after an error.**

**三、回射服务器实现框架**

……

listen();

fd\_set rset;

fd\_set allset;

FD\_ZERO(&rset);

FD\_ZERO(&allset);

FD\_SET(listenfd, &allset);

while (1) {

rset = allset;

nready = select(maxfd + 1, &rset, NULL, NULL, NULL);

……

if (FD\_ISSET(listenfd, &rset)) {

……

conn = accept(listenfd, (struct sockaddr\*)&peeraddr, &peerlen); //accept不再阻塞

处理客户端链接，主动套接字文件描述符保存在数组client[n]中

FD\_SET(conn, &allset);

}

for (i = 0; i <= maxi; i++) {

conn = client[i];

if (FD\_ISSET(conn, &rset)) {

处理链接

}

}

**}**