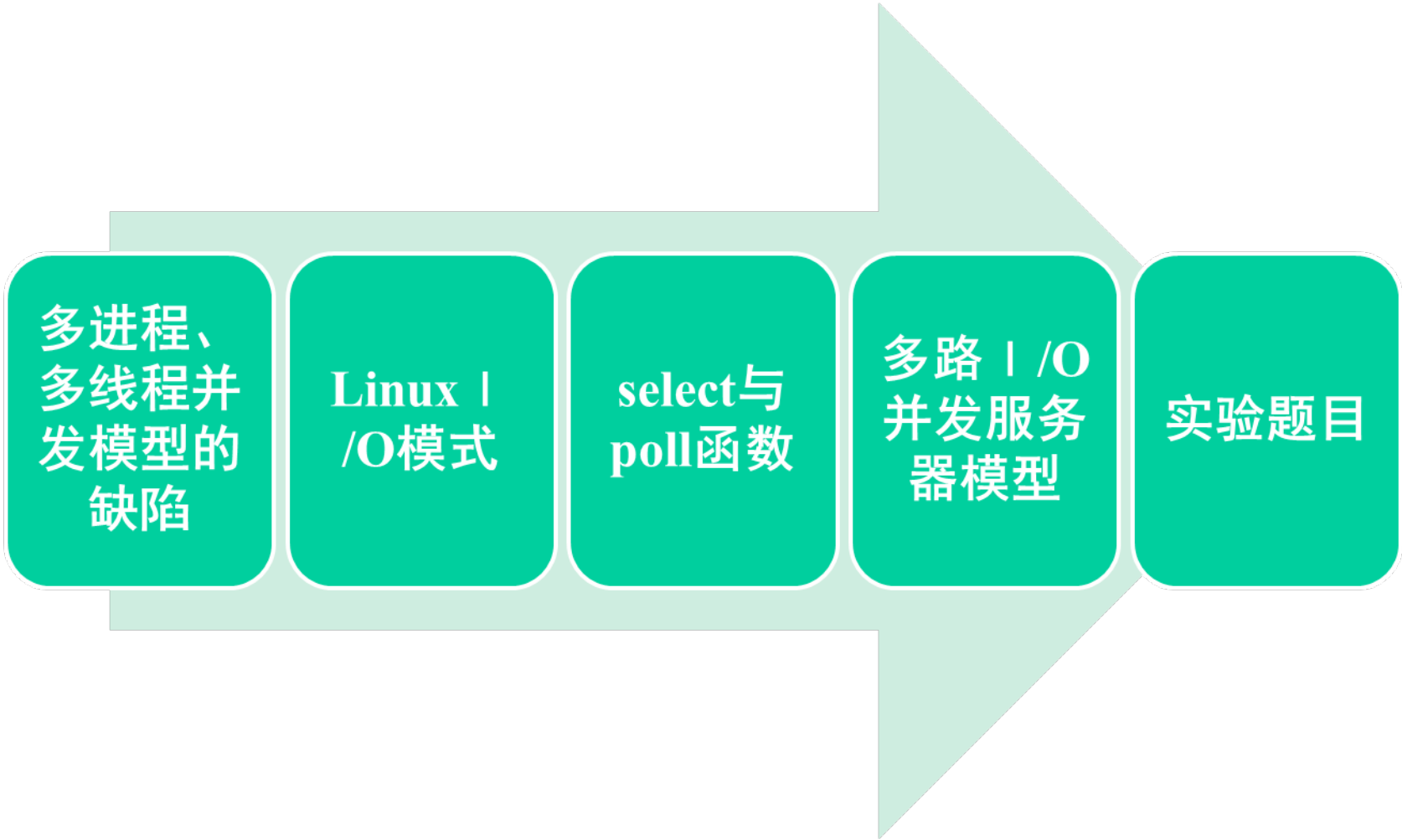

# 并发服务器 III——多路 I/O 模型

2012-11

# 课程内容


多进程、多线程并发模型的缺陷
Linux I/O模式
select与poll函数
多路I/O并发服务器模型
实验题目

# 多进程、多线程并发服务器模型的缺陷

首先，多进程socket server。这种方式对于每个请求的连接新建一个进程来处理，最大的性能损耗就是创建进程和销毁进程的损耗，在长连接，新建连接数不多的情况下，这个损耗无足轻重，但在短连接，新建连接数非常多的情况下，这个损耗就变得不可接受，web服务器是短连接的典型。

一个可以采取的优化方法是使用多线程来代替多进程。通常我们认为，创建线程的开销要低于创建进程的开销，因此，使用线程来代替进程可以降低一些性能损耗，但这并不是根本解决方案。

# 解决方法

线程池

多路I/O 复用

# 输入输出操作

输入输出（I/O）操作是在主存和外部设备（磁盘、终端、网络）之间拷贝数据的过程。输入时从I/O设备拷贝数据到主存，而输出操作则是从主存拷贝数据到终端设备。

# 输入输出模式

输入：
1．等待外设有数据可以读；
2．将数据从系统内核中拷贝到程序的数据区。

输出：
1、等待外设有足够的缓冲区；
2、将数据从内核缓冲区拷贝到外设缓冲区；

对于一个对套接字的输入操作，第一步一般来说是等待数据从网络上传到本地。当数据包到达的时候，数据将会从网络层拷贝到内核的缓存中；第二步是从内核中把数据拷贝到程序的数据区中。

# Linux I/O 工作模式

I/O的方式有很多种，我们之前使用的IO方式的特点是：

- 单路：只能等待一个fd可读或可写
- 阻塞：睡眠直到fd可读或可写
- 同步：read和write必须结束才返回？？

因此有与之对应的：

- 多路：同时等待多个fd可读或可写
- 非阻塞：fd不可读或不可写立即返回
- 异步：I/O没有结束read和write也可返回

# Linux I/O 模式

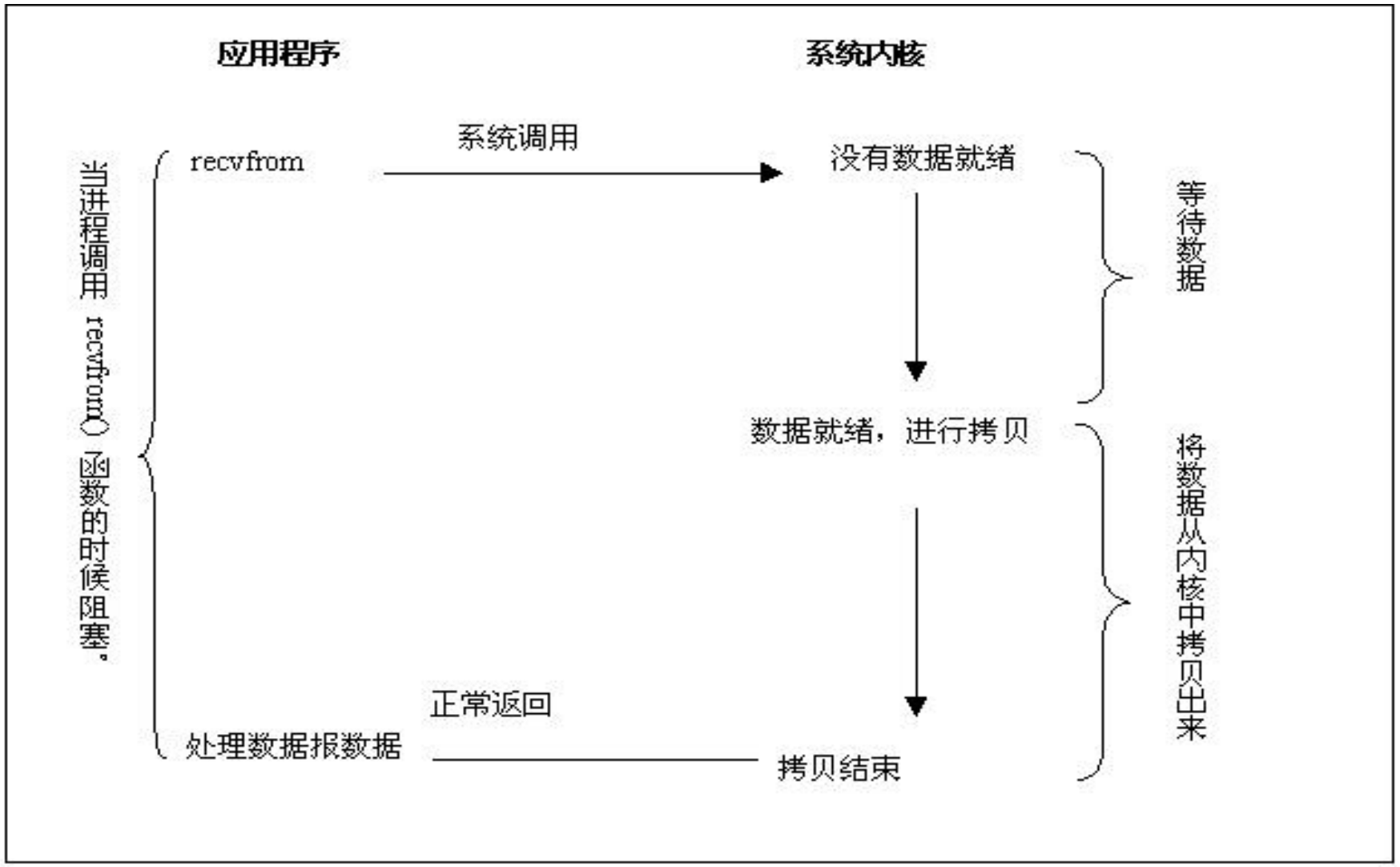
- 阻塞模式
- 非阻塞模式
- I/O多路复用
- 信号驱动I/O
- 异步I/O

# 阻塞模式

1、阻塞方式是我们最常用的I/O方式，函数库中的大部分函数都是以阻塞模式对I/O进行操作的。

2、若进程对外设进行操作，外设不满足条件时，进程的操作将被阻塞，直到响应的条件满足，操作才返回。

# 阻塞模式


应用程序
系统内核
系统调用
recvfrom
没有数据就绪
等待数据
数据就绪，进行拷贝
将数据从内核中拷贝出来
正常返回
处理数据报数据
拷贝结束
当进程调用 recvfrom() 函数的时候阻塞。

# 阻塞模式特点

这种模式简单，程序员容易理解，因此在应用程序特别是网络程序中得到广发的使用；

程序在I/O操作时，宝贵的CPU资源被浪费掉；

例子：
1.客户端通过 recv 从缓冲区读取数据时，服务器异常终止，造成客户端 read 程序被一直阻塞；
2.在多进程 / 多线程服务器模型中，当其中一个进程或线程被一个套接字阻塞，造成进程处于睡眠状态，因而不能处理其它套接字的内容；

# 非阻塞方式

•非阻塞方式输入输出是指无论你外设的条件是否满足，进程都不阻塞，若有数据，返回正常的读写字节数，若没有数据，则函数立即以错误方式返回。

•读：

进程调用 read 操作读取数据，如果缓冲区中没有数据， read 函数将立即返回错误，错误类型为 EWOULDBLOCK ，表示函数本该阻塞，但由于 I/O 处于非阻塞方式，所以调用立即返回。如果缓冲区数据小于进程想要的数据，则 read 返回实际读取的字节数；

•写

进程调用 write 操作写数据，如果没有空闲的缓冲区中， write 函数将立即返回错误，错误类型为 EWOULDBLOCK ，表示函数本该阻塞，但由于 I/O 处于非阻塞方式，所以调用立即返回。如果缓冲区有部分空闲空间，但不足以存放全部数据， write 则只将前面的部分数据放到缓冲区中，返回实际写的字节数；

# 非阻塞方式的实现

可以通过两个函数改变 I/O 工作模式

- 函数 fcntl 可以通过设置一个文件描述的标志为 O_NONBLOCK 来实现非阻塞；
- 通过函数 ioctl 使用 FIONBIO 参数可以将一个文件描述符设置为非阻塞方式；

# 非阻塞方式的特点

进程采用非阻塞方式，进程的效率得以提高；

当一个应用程序使用了非阻塞模式的套接字，它需要使用一个循环来不听的测试是否一个文件描述符有数据可读（称做polling）。应用程序不停的polling 内核来检查是否I/O操作已经就绪。这将是一个极浪费CPU 资源的操作。这种模式使用中不是很普遍。

# 多路 I/O 复用

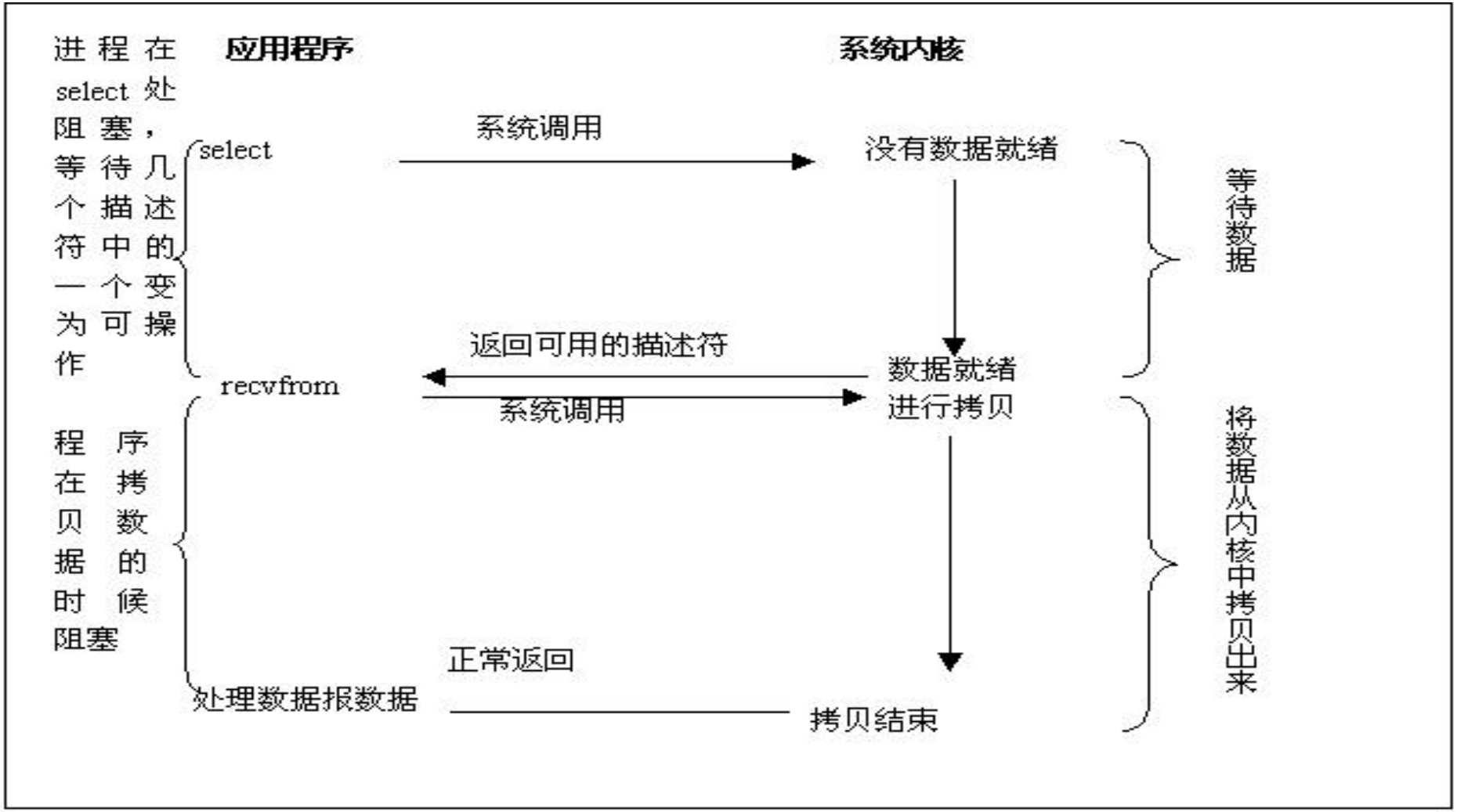
定义： **I/O** 多路复用：我们在操作 **I/O** 时，如果一个或多个 **I/O** 条件满足时，我们可以被通知到的能力。

在使用 I/O 多路技术的时候，我们调用 select() 函数和 poll() 函数，在调用它们的时候阻塞，而不是我们来调用的 IO 函数比如 recvfrom （或 recv ）的时候阻塞。当我们调用 select 函数阻塞的时候， select 函数等待数据报套接字进入读就绪状态。当 select 函数返回的时候，也就是套接字可以读取数据的时候。这时候我们就可以调用 recvfrom 函数来将数据拷贝到我们的程序缓冲区中。和阻塞模式相比较，

# 多路 I/O 复用

select() 和 poll() 并没有什么高级的地方，而且，在阻塞模式下只需要调用一个函数：读取或发送，在使用了多路复用技术后，我们需要调用两个函数了：先调用 select() 函数或 poll() 函数，然后才能进行真正的读写。多路复用的高级之处在于，它能同时等待多个文件描述符，而这些文件描述符（套接字描述符）其中的任意一个进入读就绪状态， select() 函数就可以返回。

# 多路 I/O 复用工作过程


进程在select处阻塞，等待几个描述符中的一个变为可操作
应用程序
系统内核
select
系统调用
没有数据就绪
等待数据
返回可用的描述符
recvfrom
数据就绪
进行拷贝
系统调用
程序在拷贝数据的时候阻塞
将数据从内核中拷贝出来
正常返回
处理数据报数据
拷贝结束

# 多路复用函数 select

```
#include <sys/time.h>
int select(int numfds, fd_set *readfds, fd_set *writefds,
fd_set *exceptfds, struct timeval *timeout);
```

参数说明

1. numfds 是 readfds ， writefds ， exceptfds 中 fd 集合中文件描述符中最大的数字加上
2. readfds 中的 fd 集合将由 select 来监视是否可以读取。
3. writefds 中的 fds 集合将由 select 来监视是否可以写入。
4. exceptfds 中的 fds 集合将由 select 来监视是否有例外发生。

# 文件描述符集合操作函数

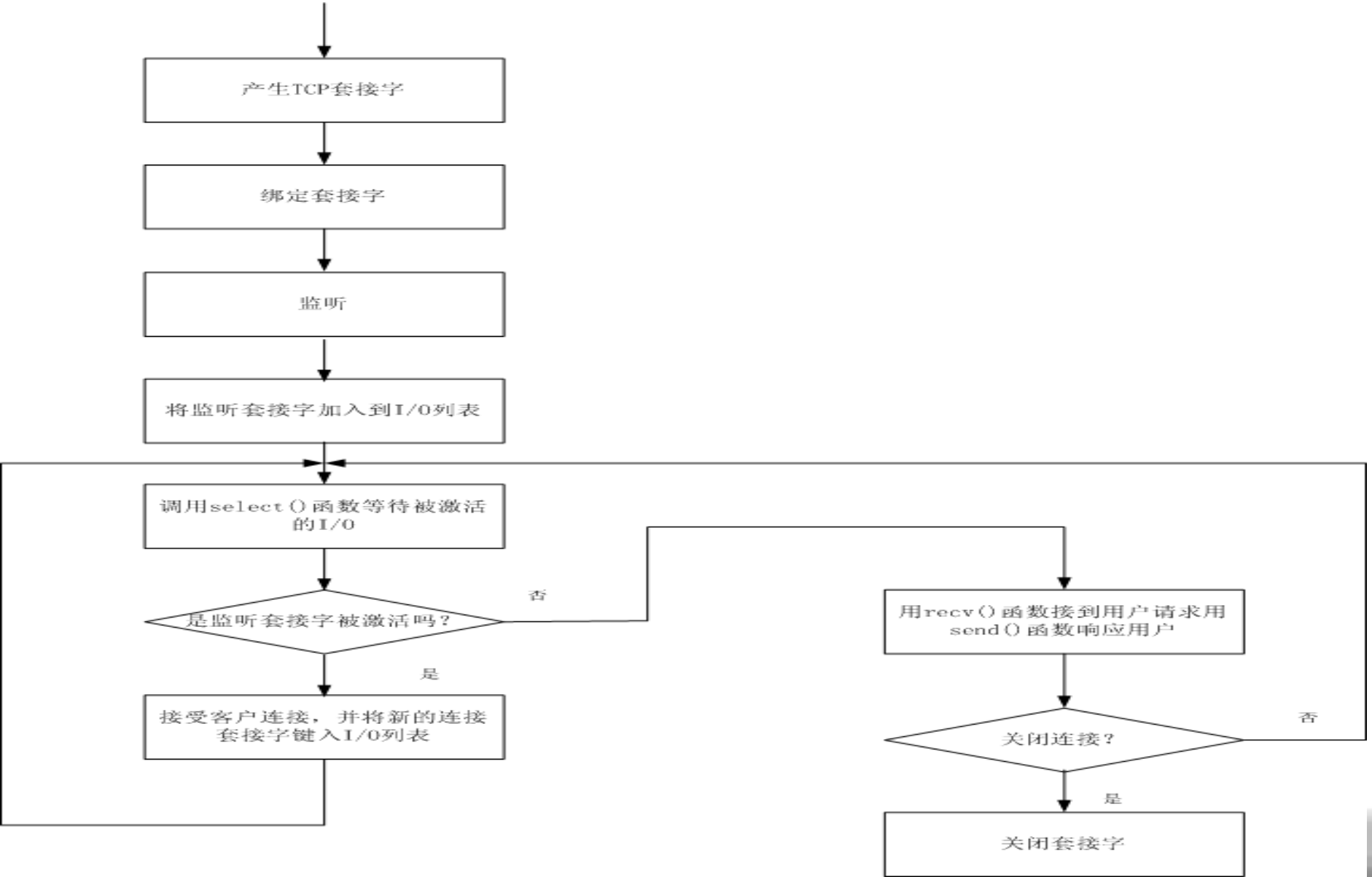
1. FD_ZERO （ fd_set *set ）将一个文件描述符集合清零
2. FD_SET （ int fd, fd_set *set ）将文件描述符 fd 加入集合 set 中。
3. FD_CLR （ int fd, fd_set *set ）将文件描述符 fd 从集合 set 中删除 .
4. FD_ISSET （ int fd, fd_set *set ）测试文件描述符 fd 是否存在于文件描述符 set 中 .

# 时间参数

```
struct timeval
{
int tv_sec ; /*  秒数  */
int tv_usec ; /*  微秒  */
} ;
```

只需要将 tv_sec 设置为你想等待的秒数，然后设置 tv_usec 为想等待的微秒数（真正的时间就是 tv_sec 所表示的秒数加上 tv_usec 所表示的微秒数）．注意，是微秒（百万分之一）而不是毫秒．一秒有 1,000 毫秒，一毫秒有 1,000 微秒。所以，一秒有 1,000,000 微秒．当 select() 函数返回的时候， timeval 中的时间将会被设置为执行为 select() 后还剩下的时间。

# 多路 I/O 多路复用并发服务器模型



# 实验题目

结合教师给的两个参考程序（基于 select 的远程并发服务器和基于 select 的聊天程序），利用 select 函数实现在 Linux 环境下实现一个聊天室程序，要求：

1. 用户默认处于广播模式，一个客户在其客户端发送的消息，其它客户端用户全部可以收到；
2. 程序支持下列命令

   /help: 显示帮助信息（思考：信息是放在客户端还是服务器端）；

   /quit: 用户退出聊天室，同时将退出信息广播给其他用户；

   /who: 显示在线用户；

   /send 用户名 消息：向指定用户发送点到点消息

# 思路

服务器：

1、完成服务器的初始化工作；

2、对侦听套接字进行侦听；

3、将侦听套接字放入读集合；

4、 while(1)

{

select(maxfd+1,读套接字, …

) ;

如果是侦听套接字被激活

{

建立连接套接字；

将连接套接字放入读套接字集合

}

如果是连接套接字被激活

{

利用连接套接字进行数据通讯；

}

}

# 思路

客户器端：

1、套接字初始化；

2、调用 connect 建立连接；

3、服务器发送客户名字；

4、将套接字和标准输入加入读集合

5、 while(1)

{

如果是套接字被激活

将数据通过套接字发送给服务器；

}

如果是标准输入被激活

{ 将数据放入发送缓冲区；

判断输入是否为“/quit”

如果是，退出客户端 程序；

}

}

# 数据处理的思路（１）

```
typedef strcut client
{
    int fd;// 客户端连接套接字
    char usrname[256];// 客户的名字
    int first;// 用于只是用户是否刚刚登陆
}
```

# 数据处理的思路（2）

1、接受客户端的输入；

2、判断客户是否是首次登陆；（根据 first）

3、如果是首次登陆，则将用户的数据登记到服务器 main 函数的局部变量 client[FD_SETSIZE] 中；

4、不是，判断信息是否为命令（首字符为‘/’），不是，将消息广播发送，是的话，对命令解析执行；

# 广播的工作原理

```
For(count=0;count<FD_SIZE;count++)
{
  if (-1==cli[count].fd) continue;
   调用 send 发送数据；

}
```

# 显示在线用户

```
For (count=0;count<   FD_SETSIZE;count++)
{
  if (-1==cli[count].fd) continue;
   将用户名放到缓冲区；

}
```

以点到点的方式发动到对应的客户端

# 点到点通信

1、解析用户的名字，将接受的数据分别存放到 cmd,usrname 和 message 三个缓冲区；

2、判断 cmd 是否为 send，否的话在对应客户端打印帮助信息；

3、在 cli[FD_SETSIZE] 中查询用户 usrname，如存在，则使用用户名对应的套接字信息传送给指定用户，如果不存在，则将“用户不存在”信息显示给发起端；