作者：linux技术栈  
链接：https://zhuanlan.zhihu.com/p/379708228  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

在TCP[并发编程](https://www.zhihu.com/search?q=%E5%B9%B6%E5%8F%91%E7%BC%96%E7%A8%8B&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A%22379708228%22%7D)中，通常使用**one loop per thread**的并发模型，也就是使用多个线程，每个线程中都有一个epoll loop，无论是使用epoll还是poll或select，在观察有无数据就绪时，都是针对多个文件描述符。如果只有一个[文件描述符](https://www.zhihu.com/search?q=%E6%96%87%E4%BB%B6%E6%8F%8F%E8%BF%B0%E7%AC%A6&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A%22379708228%22%7D)，那么进程只要观察那一个文件描述符即可。

网络编程中，一个Socket对应一个文件描述符。在TCP的并发中，服务器在监听端口初始化一个socket套接字描述符，接受客户端后就与每个客户端的连接有一个不同的文件描述符，所以TCP并发中有多个socket套接字描述符。但是，UDP协议的服务器没有真正意义上的“连接”的概念。在消息监听端口和响应请求都只有一个socket套接字描述符。

**UDP怎么考虑并发？**

《UNP · 卷1》中UDP章节描述了一句话：一般来说大多数TCP服务器是并发的，而**大多数UDP服务器是迭代的**。也就是服务器等待[客户端](https://www.zhihu.com/search?q=%E5%AE%A2%E6%88%B7%E7%AB%AF&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A%22379708228%22%7D)的请求，然后读取请求后处理，再发回响应。如果是简单的处理响应还可以，如果每个处理都很耗时，那么就不得不考虑在UDP服务器做并发处理。

并发常见的思路就是[多线程](https://www.zhihu.com/search?q=%E5%A4%9A%E7%BA%BF%E7%A8%8B&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A%22379708228%22%7D)。服务器读取一个新的请求后，可以交给一个线程处理，该线程在处理之后直接将响应内容发给客户端。

虽然可以多线程处理读取的每个消息，但如果UDP服务器与多个客户端交互，却没有多个socket，这样效率并不是很高。典型的解决方法就是：在服务器为每个客户端创建一个新的socket套接字并绑定一个新的端口，客户端以后就需要以这个新的socket套接字与服务器通信。

总的来说：UDP并发服务器针对多个客户端，可以创建多个socket。针对多个请求，可以使用多线程（线程池）进行处理。

**UDP** [**并发编程模型**](https://www.zhihu.com/search?q=%E5%B9%B6%E5%8F%91%E7%BC%96%E7%A8%8B%E6%A8%A1%E5%9E%8B&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A%22379708228%22%7D)

* **多个socket**（[**伪代码**](https://www.zhihu.com/search?q=%E4%BC%AA%E4%BB%A3%E7%A0%81&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A%22379708228%22%7D)）

for (; ;)

{

//等待新的客户端连接

recvfrom(&from\_addr);

//每有一个新的客户端，创建一个线程

pthread\_create(&tid, NULL, thread\_fun, &from\_addr);

}

//线程函数

void\* thread\_fun(void\* arg)

{

peer = socket(AF\_INET, SOCK\_DGRAM, 0);

servaddr.sin\_port = htons(0); //绑定端口

bind(peer, (struct sockaddr\*)&servaddr, sizeof(servaddr));

//将这个套接字和客户端地址连接，之后就可以使用write/read或send/recv这些函数，且不用再关心客户端地址

connect(peer, (struct sockaddr\*)&from, sizeof(from));

//处理请求的loop

}

* **使用epoll进行处理**（**伪代码**）

1. UDP服务器创建socket，并设置socket为REUSEADDR、REUSEPORT和非阻塞同时再bind服务器地址local\_addr：

listen\_fd = socket(PF\_INET, SOCK\_DGRAM, 0);

fcntl(listen\_fd, F\_SETFL, fcntl(listen\_fd, F\_GETFD, 0)|O\_NONBLOCK)

setsockopt(listen\_fd, SOL\_SOCKET, SO\_REUSEADDR, &opt, sizeof(opt));

setsockopt(listen\_fd, SOL\_SOCKET, SO\_REUSEPORT, &opt, sizeof(opt));

bind(listen\_fd, (struct sockaddr \*)&local\_addr, sizeof(struct sockaddr));

2. 创建[epoll fd](https://www.zhihu.com/search?q=epoll+fd&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A%22379708228%22%7D)，并将listen\_fd添加到epoll中，并监听其可读事件：

epoll\_fd = epoll\_create(100);

ep\_event.events = EPOLLIN | EPOLLET;

ep\_event.data.fd = listen\_fd;

epoll\_ctl(epoll\_fd, EPOLL\_CTL\_ADD, listen\_fd, &ep\_event);

while (1)

{

in\_fds = epoll\_wait(epoll\_fd, in\_events, 1000, 5000);

3. epoll\_wait返回时，如果返回的是listen\_fd， 调用recvfrom接受client第一个UDP包，并根据recvfrom返回client地址，创建一个新的socket套接字new\_fd，设置new\_fd为REUSEADDR、REUSEPORT和非阻塞，同时bind本地地址local\_addr然后connect上recvfrom返回的client地址：

for (i = 0; i < in\_fds; i++)

{

if(in\_events[i].data.fd = listen\_fd)

{

recvfrom(listen\_fd, buf, sizeof(buf), 0, (struct sockaddr \*）&client\_addr, &client\_len);

new\_fd = socket(PF\_INET, SOCK\_DGRAM, 0);

fcntl(new\_fd, F\_SETFL, fcntl(new\_fd, F\_GETFD, 0)|O\_NONBLOCK);

setsockopt(new\_fd, SOL\_SOCKET, SO\_REUSEADDR, &opt, sizeof(opt));

setsockopt(new\_fd, SOL\_SOCKET, SO\_REUSEPORT, &opt, sizeof(opt));

bind(new\_fd, (struct sockaddr \*)&local\_addr, sizeof(struct sockaddr));

connect(new\_fd, (struct sockaddr \*)&client\_addr, sizeof(struct sockaddr));

4. 将新创建的new\_fd加入到epoll中并监听其可读事件：

client\_ev.events = EPOLLIN;

client\_ev.data.fd = new\_fd;

epoll\_ctl(epoll\_fd, EPOLL\_CTL\_ADD, new\_fd, &client\_ev);

}

else if (in\_events[i].events & EPOLLIN)

{

5. 当epoll\_wait返回时，如果返回的是new\_fd，那么调用recvfrom来接收特定client的UDP包：

recvfrom(new\_fd, recvbuf, sizeof(recvbuf), 0, (struct sockaddr \*)&client\_addr, &client\_len);

data->fd = new\_fd;

data-> ptr= process(recvbuf); /\*data中包括socket信息\*/

ev.data.ptr = data;

ev.events = EPOLLOUT | EPOLLET;

epoll\_ctl(epoll\_fd,EPOLL\_CTL\_MOD,new\_fd,&ev);

}

else if (in\_events[i].events & EPOLLOUT)

{

sockfd = data->fd;

send( sockfd, data->ptr, strlen((char\*)data->ptr), 0 );

ev.data.ptr = data;

ev.events = EPOLLIN | EPOLLET;

epoll\_ctl(epoll\_fd,EPOLL\_CTL\_MOD,sockfd,&ev);

}

else

{

}

}

}

Linux服务端开发学习资料（资料包括C/C++，Linux，golang技术，Nginx，ZeroMQ，MySQL，Redis，fastdfs，MongoDB，ZK，流媒体，CDN，P2P，K8S，Docker，TCP/IP，协程，DPDK，ffmpeg等），免费分享有需要的可以自行添加学习交流[群960994558](https://link.zhihu.com/?target=https%3A//jq.qq.com/%3F_wv%3D1027%26k%3Dbrzfc7hj)