<https://blog.csdn.net/kongxian2007/article/details/49153801>

我们先来看结果:

Client:

**[cpp]** [view plain](https://blog.csdn.net/kongxian2007/article/details/49153801) [copy](https://blog.csdn.net/kongxian2007/article/details/49153801)

1. #include <sys/socket.h>
2. #include <stdio.h>
3. #include <string.h>
4. #include <netinet/in.h>
5. #include <stdlib.h>
6. #include <arpa/inet.h>
8. **int** main(**int** argc,**char**\*\* argv)
9. {
10. **int** ret;
11. **int** sockfd = socket(AF\_INET,SOCK\_STREAM,0);
12. **if** (sockfd == -1)
13. {
14. printf("socket error\n");
15. **return** -1;
16. }
18. **struct** sockaddr\_in serveraddr;
19. memset(&serveraddr,0,**sizeof**(serveraddr));
20. serveraddr.sin\_family = AF\_INET;
21. inet\_aton(argv[1],&serveraddr.sin\_addr);
22. serveraddr.sin\_port = htons((unsigned **short**)(atoi(argv[2])));
24. ret = connect(sockfd,(**const** sockaddr\*)&serveraddr,**sizeof**(serveraddr));
25. **if** (ret == -1)
26. {
27. printf("connect error,ret = %d\n",ret);
28. **return** -1;
29. }
31. **for**(;;)
32. {}
33. **return** 0;
34. }

Server:

**[cpp]** [view plain](https://blog.csdn.net/kongxian2007/article/details/49153801) [copy](https://blog.csdn.net/kongxian2007/article/details/49153801)

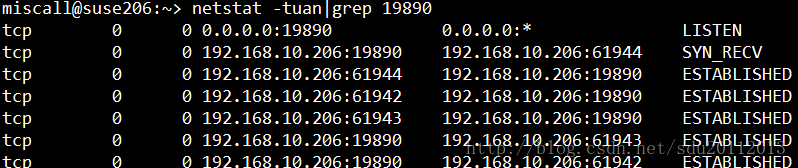
1. #include <sys/socket.h>
2. #include <stdio.h>
3. #include <string.h>
4. #include <netinet/in.h>
5. #include <stdlib.h>
6. #include <arpa/inet.h>

9. **int** main(**int** argc,**char**\*\* argv)
10. {
11. **int** ret;
12. **int** listenfd = socket(AF\_INET,SOCK\_STREAM,0);
13. **if** (listenfd == -1)
14. {
15. printf("socket error\n");
16. **return** -1;
17. }
19. **struct** sockaddr\_in serveraddr;
20. memset(&serveraddr,0,**sizeof**(serveraddr));
21. serveraddr.sin\_family = AF\_INET;
22. serveraddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);
23. serveraddr.sin\_port = htons((unsigned **short**)(atoi(argv[1])));
25. ret = bind(listenfd,(**const** sockaddr\*)&serveraddr,**sizeof**(serveraddr));
26. **if** (ret == -1)
27. {
28. printf("bind error,ret = %d\n",ret);
29. **return** -1;
30. }
32. <span style="WHITE-SPACE: pre"> </span>**int** backlog = atoi(argv[2]);
33. ret = listen(listenfd, backlog);
34. printf("backlog = %d,ret =%d\n",backlog,ret);
35. **if** (ret == -1)
36. {
37. printf("listen error,ret = %d\n",ret);
38. **return** -1;
39. }
41. **for**(;;)
42. {}
43. **return** 0;
44. }

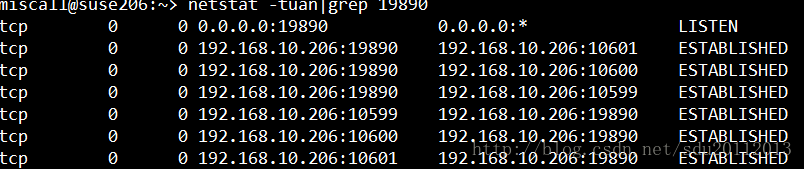
server的代码中通过命令行的方式传入backlog的值,以便于我们通过修改backlog来观察不同的backlog对程序的影响.

开启一个server与3个client.

backlog = 1时：

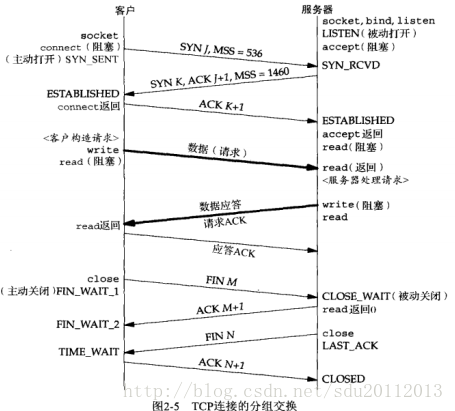


backlog = 2时：



观察上图,server在19890端口监听,3个client发起TCP连接。在第一幅图中,对client而言，3个状态均为established.对server而言,2个连接状态为established，一个为syn\_recv.第二幅图中,对server而言,3个链接的状态均为established。

我们先来看一下TCP连接的状态转换



再说回listen(int fd,int backlog)函数，内核会为一个监听套接字维护两个队列,当有TCP请求到来时,即3次握手中的syn分节发送来时,会在未完成队列中增加一项,3次握手完成时,未完成队列中的项就移动到已完成队列里,accept()函数会从已连接队列里取走已完成连接.

这个backlog参数就是控制我们的已连接队列里等待accept()取走的连接的最大数目的.注意一点，backlog与这个已排队连接的最大数目未必是完全相等的,不同的系统的实现可能不同.比如backlog=1，系统允许的实际一排队数目可能为2.

所以在上图中，对client而言，connect()返回，TCP的状态就转变为established。而对server而言,3次握手完成TCP状态才会转换为established。在图一中，由于backlog参数的限制,导致有一条连接并未完成3次握手，所以状态为syn\_recv。