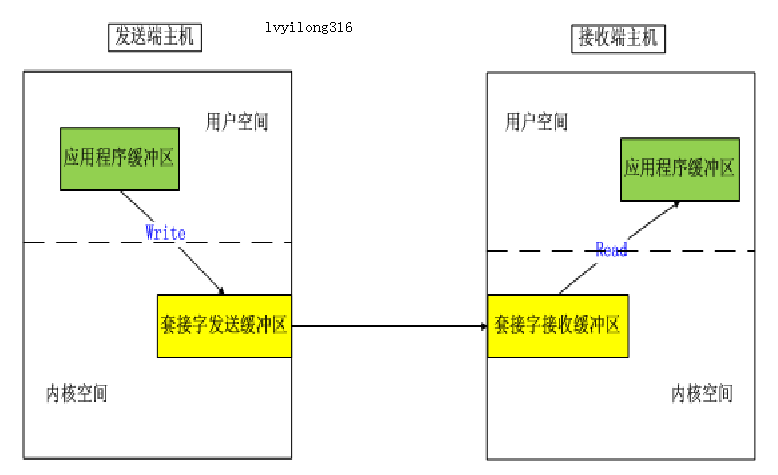
<http://blog.csdn.net/gogokongyin/article/details/51210522>

问题：**“一个tcp服务端和一个tcp客户端，客户端和服务端建立连接后，服务端一直sleep，然后客户端一直发送数据会是什么现象”。**

回答这个问题前我们先想一想tcp的特征和tcp发送数据的大体过程：

首先，tcp是有链接的可靠传输协议，所谓可靠也就是说保证客户端发送的数据服务端都能够收到，并且是按序收到。那么对于上面的问题就不可能存在数据的丢弃。那么客户端一直发送数据越来越多怎么办？下面我们分析一下tcp的传输过程。

  
图1

如图1所示当发送数据时：

(1) 数据首先由**应用程序缓冲区**复制到发送端的**套接字发送缓冲区（位于内核）**，注意这个过程是用类似write功能的函数完成的。有的人通常看到write成功就以为数据发送到了对端主机，其实这是错误的，write成功仅仅表示数据成功的由应用进程缓冲区复制到了套接字发送缓冲区。

(2) 然后内核协议栈将**套接字发送缓冲区**中的数据发送到对端主机，注意这个过程不受应用程序控制，而是发送端内核协议栈完成，其中包括使用滑动窗口、用赛控制等功能。

(3) 数据到达接收端主机的套**接字接收缓冲区**，注意这个接收过程也不受应用程序控制，而是由接收端内核协议栈完成，其中包括发送ack确认等。

(4) 数据由**套接字接收缓冲区**复制到接收端**应用程序缓冲区**，注意这个过程是由类似read等函数来完成。

**1.1 阻塞方式的情况**

知道了这个过程，我们在看一下在默认情况下（套接字为阻塞方式）write等函数的工作方式：**输出操作，包括write、writev、send、sendto和sendmsg共5个函数。对于一个TCP套接字，内核从应用进程的缓冲区到套接字的发送缓冲区复制数据。对于阻塞的套接字，如果其发送缓冲区中没有空间，进程将被投入睡眠，直到有空间为止。——UNPv1**

这样我们就可以推测出了结果：阻塞方式下，如果服务端一直sleep不接收数据，而客户端一直write，也就是只能执行上述过程中的前三步，这样最终结果肯定是接收端的套接字接收缓冲区和发送端套接字发送缓冲区都被填满，这样write就无法继续将数据从应用程序复制到发送端的套接字发送缓冲区了，从而使进程进入睡眠。

验证例子如下。

客户端代码：

**tcpClient.c**

点击(此处)折叠或打开

1. #include <stdio.h>
2. #include <string.h>
3. #include <unistd.h>
4. #include <sys/types.h>
5. #include <sys/socket.h>
6. #include <stdlib.h>
7. #include <memory.h>
8. #include <arpa/inet.h>
9. #include <netinet/in.h>
10. #define PORT 9999
11. #define Buflen 1024
12. int main(int argc,char \*argv[])
13. {
14. struct sockaddr\_in server\_addr;
15. int n,count=0;
16. int sockfd;
17. char sendline[Buflen];
18. sockfd= socket(AF\_INET,SOCK\_STREAM,0);
19. memset(&server\_addr,0,sizeof(server\_addr));
20. server\_addr.sin\_family = AF\_INET;
21. server\_addr.sin\_port = htons(PORT);
22. server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);
23. server\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]);
24. connect(sockfd,(struct sockaddr \*)&server\_addr,sizeof(server\_addr));
26. //与服务器端进行通信
27. memset(sendline,'a',sizeof(Buflen));
29. while ( (n=write(sockfd,sendline,Buflen))>0 )
30. {
31. count++;
32. printf("already write %d bytes -- %d\n",n,count);
33. }
35. if(n<0)
36. perror("write error");
37. close(sockfd);
38. }

客户端每次write成功一次，将计数器count加1，同时输出本次write成功的字节数。count保存客户端write成功的次数。

服务端代码：

**tcpServer.c**

点击(此处)折叠或打开

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <strings.h>
4. #include <sys/types.h>
5. #include <sys/socket.h>
6. #include <memory.h>
7. #include <unistd.h>
8. #include <netinet/in.h>
9. #include <arpa/inet.h>
10. #include <string.h>
11. #define PORT 9999 //定义通信端口
12. #define BACKLOG 5 //定义侦听队列长度
13. #define buflen 1024
14. int listenfd,connfd;
15. int main(int argc,char \*argv[])
16. {
17. struct sockaddr\_in server\_addr; //存储服务器端socket地址结构
18. struct sockaddr\_in client\_addr; //存储客户端 socket地址结构
19. pid\_t pid;
20. listenfd = socket(AF\_INET,SOCK\_STREAM,0);
21. memset(&server\_addr,0,sizeof(server\_addr));
22. server\_addr.sin\_family = AF\_INET; //协议族
23. server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY); //本地地址
24. server\_addr.sin\_port = htons(PORT);
25. bind(listenfd,(struct sockaddr \*)&server\_addr,sizeof(server\_addr));
26. listen(listenfd,BACKLOG);
27. for(;;)
28. {
29. socklen\_t addrlen = sizeof(client\_addr);
30. connfd = accept(listenfd,(struct sockaddr \*)&client\_addr,&addrlen);
31. if(connfd<0)
32. perror("accept error");
33. printf("receive connection\n");
34. if((pid = fork()) == 0)
35. {
36. close(listenfd);
37. sleep(1000);//子进程不接收数据，sleep 1000秒
38. exit(0);
39. }
40. else
41. {
42. close(connfd);
43. }
44. }
45. }

首先编译运行服务端，然后启动客户端，运行结果如图2所示。

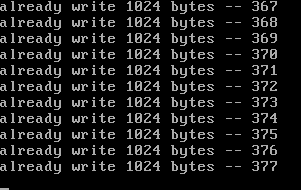


图2

可以看到客户端write成功377次后就陷入了阻塞，**注意这个时候不能说明发送端的套接字发送缓冲区一点是满的，只能说明套接字发送缓冲区的可用空间小于write请求写的自己数——1024。**

**补充：**当服务端sleep到1000后，会关闭当前连接，此时客户端处于阻塞中的write会返回错误，效果如图3。

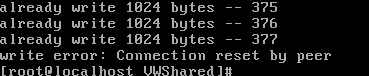


图3

**1.2 非阻塞方式的情况**

下面看一下非阻塞套接字情况下，write的工作方式：**对于一个非阻塞的TCP套接字，如果发送缓冲区中根本没用空间，输出函数将立即返回一个EWOULDBLOCK错误。如果发送缓冲区中有一些空间，返回值将是内核能够复制到该缓冲区的字节数。这个字节数也成为“不足计数”。——UNPv1**

这样就可以知道非阻塞情况下服务端一直sleep，客户端一直write数据的效果了：开始客户端write成功，随着客户端write，接收端的套接字接收缓冲区和发送端的套接字发送缓冲区会被填满。当发送端的套接字发送缓冲区的可用空间小于write请求写的字节数时，write立即返回-1，并将errno置为**EWOULDBLOCK**。

验证例子代码如下。

服务端同阻塞情况（略）。

客户端（非阻塞模式）：

**tcpClientNonBlock.c**

点击(此处)折叠或打开

1. #include <stdio.h>
2. #include <string.h>
3. #include <unistd.h>
4. #include <sys/types.h>
5. #include <sys/socket.h>
6. #include <stdlib.h>
7. #include <memory.h>
8. #include <arpa/inet.h>
9. #include <netinet/in.h>
10. #include <fcntl.h>
11. #include <errno.h>
12. #define PORT 9999
13. #define Buflen 1024
14. int main(int argc,char \*argv[])
15. {
16. struct sockaddr\_in server\_addr;
17. int n,flags,count=0;
18. int sockfd;
19. char sendline[Buflen];
20. sockfd= socket(AF\_INET,SOCK\_STREAM,0);
21. memset(&server\_addr,0,sizeof(server\_addr));
22. server\_addr.sin\_family = AF\_INET;
23. server\_addr.sin\_port = htons(PORT);
24. server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);
25. server\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]);
26. connect(sockfd,(struct sockaddr \*)&server\_addr,sizeof(server\_addr));
27. flags=fcntl(sockfd,F\_GETFL,0); //将已连接的套接字设置为非阻塞模式
28. fcntl(sockfd,F\_SETFL,flags|O\_NONBLOCK);
29. memset(sendline,'a',sizeof(Buflen));
31. while ( (n=write(sockfd,sendline,Buflen))>0 )
32. {
33. count++;
34. printf("already write %d bytes -- %d\n",n,count);
35. }
37. if(n<0)
38. {
39. if(errno!=EWOULDBLOCK)
40. perror("write error");
41. else
42. printf("EWOULDBLOCK ERROR\n");
43. }
44. close(sockfd);
45. }

首先编译运行服务端，然后启动客户端，运行结果如下图4所示。

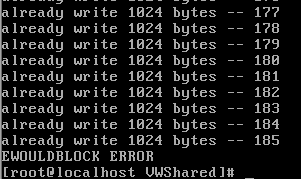


图4

可以看到客户端成功write 185次后就发生套接字发送缓冲区空间不足，从而返回EWOULDBLOCK错误。我们注意到每次write同样的字节数(1024)阻塞模式下能write成功377次，为什么非阻塞情况下要少呢？这是因为阻塞模式下一直write到接收端的**套接字接收缓冲区**和发送端的**套接字发送缓冲区**都满的情况才会阻塞。而非阻塞模式情况下有可能是发送端发送过程的第二步较慢，造成发送端的套接字发送缓冲区很快写满，而接收端的套接字接收缓冲区还没有满，这样write就会仅仅因为发送端的套接字发送缓冲区满而返回错误(准确的说的套接字发送缓冲区的可用空间小于write请求写的字节数)。对比一下377正好是185的二倍左右，所以可以推测由于发送过程第二步的延迟，很可能发送端的套接字发送缓冲区已经满了，而接收端的套接字接收缓冲区还是空的。

**1.3 UDP情况补充**

**对于UDP套接字不存在真正的发送缓冲区。内核只是复制应用进程数据并把它沿协议栈向下传送，渐次冠以UDP首部和IP首部。因此对于一个阻塞的UDP套接字（默认设置），输出函数调用将不会因为与TCP套接字一样的原因而阻塞，不过有可能会因为其他原因而阻塞。——UNPv1**

转载地址：

http://blog.chinaunix.net/xmlrpc.php?r=blog/article&uid=28541347&id=4730278

http://ticktick.blog.51cto.com/823160/779866

感谢博主的分享！