[Linux进程间通信——使用流套接字](http://blog.csdn.net/ljianhui/article/details/10477427)

前面说到的进程间的通信，所通信的进程都是在同一台计算机上的，而使用socket进行通信的进程可以是同一台计算机的进程，也是可以是通过网络连接起来的不同计算机上的进程。通常我们使用socket进行网络编程，这里将会简单地讲述如何使用socket进行简单的网络编程。

一、什么是socket

socket，即套接字是一种通信机制，凭借这种机制，客户/服务器（即要进行通信的进程）系统的开发工作既可以在本地单机上进行，也可以跨网络进行。也就是说它可以让不在同一台计算机但通过网络连接计算机上的进程进行通信。也因为这样，套接字明确地将客户端和服务器区分开来。

二、套接字的属性

套接字的特性由3个属性确定，它们分别是：域、类型和协议。

1、套接字的域

它指定套接字通信中使用的网络介质，最常见的套接字域是AF\_INET，它指的是Internet网络。当客户使用套接字进行跨网络的连接时，它就需要用到服务器计算机的IP地址和端口来指定一台联网机器上的某个特定服务，所以在使用socket作为通信的终点，服务器应用程序必须在开始通信之前绑定一个端口，服务器在指定的端口等待客户的连接。另一个域AF\_UNIX表示UNIX文件系统，它就是文件输入/输出，而它的地址就是文件名。

2、套接字类型

因特网提供了两种通信机制：流（stream）和数据报（datagram），因而套接字的类型也就分为流套接字和数据报套接字。这里主要讲流套接字。

流套接字由类型SOCK\_STREAM指定，它们是在AF\_INET域中通过TCP/IP连接实现，同时也是AF\_UNIX中常用的套接字类型。流套接字提供的是一个有序、可靠、双向字节流的连接，因此发送的数据可以确保不会丢失、重复或乱序到达，而且它还有一定的出错后重新发送的机制。

与流套接字相对的是由类型SOCK\_DGRAM指定的数据报套接字，它不需要建立连接和维持一个连接，它们在AF\_INET中通常是通过UDP/IP协议实现的。它对可以发送的数据的长度有限制，数据报作为一个单独的网络消息被传输,它可能会丢失、复制或错乱到达，UDP不是一个可靠的协议，但是它的速度比较高，因为它并一需要总是要建立和维持一个连接。

3、套接字协议

只要底层的传输机制允许不止一个协议来提供要求的套接字类型，我们就可以为套接字选择一个特定的协议。通常只需要使用默认值。

三、套接字地址

每个套接字都有其自己的地址格式，对于AF\_UNIX域套接字来说，它的地址由结构sockaddr\_un来描述，该结构定义在头文件sys/un.h中，它的定义如下：

**[cpp]** [view plain](http://blog.csdn.net/ljianhui/article/details/10477427) [copy](http://blog.csdn.net/ljianhui/article/details/10477427)

[print?](http://blog.csdn.net/ljianhui/article/details/10477427)[在CODE上查看代码片](https://code.csdn.net/snippets/669568)

1. **struct** sockaddr\_un{
2. sa\_family\_t sun\_family;//AF\_UNIX，它是一个短整型
3. **char**        sum\_path[];//路径名
4. };

对于AF\_INET域套接字来说，它的地址结构由sockaddr\_in来描述，它至少包括以下几个成员：

**[cpp]** [view plain](http://blog.csdn.net/ljianhui/article/details/10477427) [copy](http://blog.csdn.net/ljianhui/article/details/10477427)

[print?](http://blog.csdn.net/ljianhui/article/details/10477427)[在CODE上查看代码片](https://code.csdn.net/snippets/669568)

1. **struct** sockaddr\_in{
2. **short** **int**            sin\_family;//AF\_INET
3. unsigned **short** **int**    sin\_port;//端口号
4. **struct** in\_addr        sin\_addr;//IP地址
5. };

而in\_addr被定义为：

**[cpp]** [view plain](http://blog.csdn.net/ljianhui/article/details/10477427) [copy](http://blog.csdn.net/ljianhui/article/details/10477427)

[print?](http://blog.csdn.net/ljianhui/article/details/10477427)[在CODE上查看代码片](https://code.csdn.net/snippets/669568)

1. **struct** in\_addr{
2. unsigned **long** **int** s\_addr;
3. };

四、基于流套接字的客户/服务器的工作流程

使用socket进行进程通信的进程采用的客户/服务器系统是如何工作的呢？

1、服务器端

首先服务器应用程序用系统调用socket来创建一个套接安，它是系统分配给该服务器进程的类似文件描述符的资源，它不能与其他的进程共享。

接下来，服务器进程会给套接字起个名字，我们使用系统调用bind来给套接字命名。然后服务器进程就开始等待客户连接到这个套接字。

然后，系统调用listen来创建一个队列并将其用于存放来自客户的进入连接。

最后，服务器通过系统调用accept来接受客户的连接。它会创建一个与原有的命名套接不同的新套接字，这个套接字只用于与这个特定客户端进行通信，而命名套接字（即原先的套接字）则被保留下来继续处理来自其他客户的连接。

2、客户端

基于socket的客户端比服务器端简单，同样，客户应用程序首先调用socket来创建一个未命名的套接字，然后将服务器的命名套接字作为一个地址来调用connect与服务器建立连接。

一旦连接建立，我们就可以像使用底层的文件描述符那样用套接字来实现双向数据的通信。

五、流式socket的接口及作用

socket的接口函数声明在头文件sys/types.h和sys/socket.h中。

1、创建套接字——socket系统调用

该函数用来创建一个套接字，并返回一个描述符，该描述符可以用来访问该套接字，它的原型如下：

**[cpp]** [view plain](http://blog.csdn.net/ljianhui/article/details/10477427) [copy](http://blog.csdn.net/ljianhui/article/details/10477427)

[print?](http://blog.csdn.net/ljianhui/article/details/10477427)[在CODE上查看代码片](https://code.csdn.net/snippets/669568)

1. **int** socket(**int** domain, **int** type, **int** protocol);

函数中的三个参数分别对应前面所说的三个套接字属性。protocol参数设置为0表示使用默认协议。

2、命名（绑定）套接字——bind系统调用

该函数把通过socket调用创建的套接字命名，从而让它可以被其他进程使用。对于AF\_UNIX，调用该函数后套接字就会关联到一个文件系统路径名，对于AF\_INET，则会关联到一个IP端口号。函数原型如下：

**[cpp]** [view plain](http://blog.csdn.net/ljianhui/article/details/10477427) [copy](http://blog.csdn.net/ljianhui/article/details/10477427)

[print?](http://blog.csdn.net/ljianhui/article/details/10477427)[在CODE上查看代码片](https://code.csdn.net/snippets/669568)

1. **int** bind( **int** socket, **const** **struct** sockaddr \*address, **size\_t** address\_len);

成功时返回0，失败时返回-1；

3、创建套接字队列（监听）——listen系统调用

该函数用来创建一个队列来保存未处理的请求。成功时返回0，失败时返回-1，其原型如下：

**[cpp]** [view plain](http://blog.csdn.net/ljianhui/article/details/10477427) [copy](http://blog.csdn.net/ljianhui/article/details/10477427)

[print?](http://blog.csdn.net/ljianhui/article/details/10477427)[在CODE上查看代码片](https://code.csdn.net/snippets/669568)

1. **int** listen(**int** socket, **int** backlog);

backlog用于指定队列的长度，等待处理的进入连接的个数最多不能超过这个数字，否则往后的连接将被拒绝，导致客户的连接请求失败。调用后，程序一直会监听这个IP端口，如果有连接请求，就把它加入到这个队列中。

4、接受连接——accept系统调用

该系统调用用来等待客户建立对该套接字的连接。accept系统调用只有当客户程序试图连接到由socket参数指定的套接字上时才返回，也就是说，如果套接字队列中没有未处理的连接，accept将阻塞直到有客户建立连接为止。accept函数将创建一个新套接字来与该客户进行通信，并且返回新套接字的描述符，新套接字的类型和服务器监听套接字类型是一样的。它的原型如下：

**[cpp]** [view plain](http://blog.csdn.net/ljianhui/article/details/10477427) [copy](http://blog.csdn.net/ljianhui/article/details/10477427)

[print?](http://blog.csdn.net/ljianhui/article/details/10477427)[在CODE上查看代码片](https://code.csdn.net/snippets/669568)

1. **int** accept(**int** socket, **struct** sockaddr \*address, **size\_t** \*address\_len);

address为连接客户端的地址，参数address\_len指定客户结构的长度，如果客户地址的长度超过这个值，它将会截断。

5、请求连接——connect系统调用

该系统调用用来让客户程序通过在一个未命名套接字和服务器监听套接字之间建立连接的方法来连接到服务器。它的原型如下：

**[cpp]** [view plain](http://blog.csdn.net/ljianhui/article/details/10477427) [copy](http://blog.csdn.net/ljianhui/article/details/10477427)

[print?](http://blog.csdn.net/ljianhui/article/details/10477427)[在CODE上查看代码片](https://code.csdn.net/snippets/669568)

1. **int** connect(**int** socket, **const** **struct** sockaddr \*address, **size\_t** address\_len);

参数socket指定的套接字连接到参数addres指定的服务器套接字。成功时返回0，失败时返回-1.

6、关闭socket——close系统调用

该系统调用用来终止服务器和客户上的套接字连接，我们应该总是在连接的两端（服务器和客户）关闭套接字。

六、进程使用流式socket进行通信

下面用多个客户程序和一个服务器程序来展示进程间如何利用套接字进行通信。

sockserver.c是一个服务器程序，它首先创建套接字，然后绑定一个端口再监听套接字，忽略子进程的停止消息等，然后它进入循环，一直循环检查是否有客户连接到服务器，如果有，则调用fork创建一个子进程来处理请求。利用read系统调用来读取客户端发来的信息，利用write系统调用来向客户端发送信息。这个服务器的工作非常简单，就是把客户发过来的字符+1，再发送回给客户。

sockclient.c是一个客户程序，它同样要先创建套接，然后连接到指定IP端口服务器，如果连接成功，就用write来发送信息给服务器，再用read获取服务器处理后的信息，再输出。

服务器sockserver.c的源代码如下：

**[cpp]** [view plain](http://blog.csdn.net/ljianhui/article/details/10477427) [copy](http://blog.csdn.net/ljianhui/article/details/10477427)

[print?](http://blog.csdn.net/ljianhui/article/details/10477427)[在CODE上查看代码片](https://code.csdn.net/snippets/669568)

1. #include <unistd.h>
2. #include <sys/types.h>
3. #include <sys/socket.h>
4. #include <netinet/in.h>
5. #include <signal.h>
6. #include <stdio.h>
7. #include <stdlib.h>
9. **int** main()
10. {
11. **int** server\_sockfd = -1;
12. **int** client\_sockfd = -1;
13. **int** client\_len = 0;
14. **struct** sockaddr\_in server\_addr;
15. **struct** sockaddr\_in client\_addr;
16. //创建流套接字
17. server\_sockfd = socket(AF\_INET, SOCK\_STREAM, 0);
18. //设置服务器接收的连接地址和监听的端口
19. server\_addr.sin\_family = AF\_INET;//指定网络套接字
20. server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);//接受所有IP地址的连接
21. server\_addr.sin\_port = htons(9736);//绑定到9736端口
22. //绑定（命名）套接字
23. bind(server\_sockfd, (**struct** sockaddr\*)&server\_addr, **sizeof**(server\_addr));
24. //创建套接字队列，监听套接字
25. listen(server\_sockfd, 5);
26. //忽略子进程停止或退出信号
27. signal(SIGCHLD, SIG\_IGN);
29. **while**(1)
30. {
31. **char** ch = '\0';
32. client\_len = **sizeof**(client\_addr);
33. printf("Server waiting\n");
34. //接受连接，创建新的套接字
35. client\_sockfd = accept(server\_sockfd, (**struct** sockaddr\*)&client\_addr, &client\_len);
37. **if**(fork() == 0)
38. {
39. //子进程中，读取客户端发过来的信息，处理信息，再发送给客户端
40. read(client\_sockfd, &ch, 1);
41. sleep(5);
42. ch++;
43. write(client\_sockfd, &ch, 1);
44. close(client\_sockfd);
45. exit(0);
46. }
47. **else**
48. {
49. //父进程中，关闭套接字
50. close(client\_sockfd);
51. }
52. }
53. }

客户sockclient.c的源代码如下：

**[cpp]** [view plain](http://blog.csdn.net/ljianhui/article/details/10477427) [copy](http://blog.csdn.net/ljianhui/article/details/10477427)

[print?](http://blog.csdn.net/ljianhui/article/details/10477427)[在CODE上查看代码片](https://code.csdn.net/snippets/669568)

1. #include <unistd.h>
2. #include <sys/types.h>
3. #include <sys/socket.h>
4. #include <netinet/in.h>
5. #include <arpa/inet.h>
6. #include <stdio.h>
7. #include <stdlib.h>
9. **int** main()
10. {
11. **int** sockfd = -1;
12. **int** len = 0;
13. **struct** sockaddr\_in address;
14. **int** result;
15. **char** ch = 'A';
16. //创建流套接字
17. sockfd = socket(AF\_INET, SOCK\_STREAM, 0);
18. //设置要连接的服务器的信息
19. address.sin\_family = AF\_INET;//使用网络套接字
20. address.sin\_addr.s\_addr = inet\_addr("127.0.0.1");//服务器地址
21. address.sin\_port = htons(9736);//服务器所监听的端口
22. len = **sizeof**(address);
23. //连接到服务器
24. result = connect(sockfd, (**struct** sockaddr\*)&address, len);
26. **if**(result == -1)
27. {
28. perror("ops:client\n");
29. exit(1);
30. }
31. //发送请求给服务器
32. write(sockfd, &ch, 1);
33. //从服务器获取数据
34. read(sockfd, &ch, 1);
35. printf("char form server = %c\n", ch);
36. close(sockfd);
37. exit(0);
38. }

运行结果如下：



在本例子中，我们启动了一个服务器程序和三个客户程序，从运行的结果来看，客户端发送给服务器程序的所有请求都得到了处理，即把A变成了B。对于服务器和客户程序之间使用的read和write系统调用跟使用命名管道时阻塞的read、write系统调用一样。例如客户程序调用read时，如果服务器程序没有向指定的客户程序的socket中写入信息，则read调用会一直阻塞。

七、流式套接字给我印象

给我的感觉是流式套接字很像命名管道，但是它却可以使不在同一台计算机而通过网络连接的不同计算机上的进程进行通信，功能真是非常的强大。