[Linux进程间通信——使用数据报套接字](http://blog.csdn.net/ljianhui/article/details/10697935)

一、简单回顾——什么是数据报套接字。

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。

相对于流套接字，数据报套接字的使用更为简单，它是由类型SOCK\_DGRAM指定的，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。

二、基于流套接字的客户/服务器的工作流程

使用数据报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

与使用流套接字一样，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（监听），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。

不同的是，然后系统调用recvfrom来接收来自客户程序发送过来的数据。服务器程序对数据进行相应的处理，再通过系统调用sendto把处理后的数据发送回客户程序。

与流套接字程序相比：

1、在流套接字中的程序中，接收数据是通过系统调用read，而发送数据是通过系统调用write来实现，而在数据报套接字程序中，这是通过recvfrom和sendto调用来实现的。

2、使用数据报套接字的服务器程序并不需要listen调用来创建一个队列来存储连接，也不需要accept调用来接收连接并创建一个新的socket描述符

2、客户端

基于数据报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，与服务器一样，客户也是通过sendto和recvfrom来向服务器发送数据和从服务器程序接收数据。

与流套接字程序相比：

使用数据报套接字的客户程序并不需要使用connect系统调用来连接服务器程序，它只要在需要时向服务器所监听的IP端口发送信息和接收从服务器发送回来的数据即可。

三、数据报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/10697935) [copy](http://blog.csdn.net/ljianhui/article/details/10697935)

[print?](http://blog.csdn.net/ljianhui/article/details/10697935)

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/10697935) [copy](http://blog.csdn.net/ljianhui/article/details/10697935)

[print?](http://blog.csdn.net/ljianhui/article/details/10697935)

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、发送数据——sendto系统调用

该函数把缓冲区buffer中的信息给送给指定的IP端口的程序，原型如下：

**[cpp]** [view plain](http://blog.csdn.net/ljianhui/article/details/10697935) [copy](http://blog.csdn.net/ljianhui/article/details/10697935)

[print?](http://blog.csdn.net/ljianhui/article/details/10697935)

1. **int** sendto(**int** sockfd, **void** \*buffer, **size\_t** len, **int** flags, **struct** sockaddr \*to, socklen\_t tolen);

buffer中储存着将要发送的数据，len是buffer的长度，而flags在应用中通常被设置为0，to是要发送数据到的程序的IP端口，tolen是to参数的长度。

成功时返回发送的数据的字节数，失败时返回-1.

4、接收数据——recvfrom系统调用

该函数把发送给程序的信息储存在缓冲区buffer中，并记录数据来源的程序IP端口，原型如下：

**[cpp]** [view plain](http://blog.csdn.net/ljianhui/article/details/10697935) [copy](http://blog.csdn.net/ljianhui/article/details/10697935)

[print?](http://blog.csdn.net/ljianhui/article/details/10697935)

1. **int** recvfrom(**int** sockfd, **void** \*buffer, **size\_t** len，**int** flags, **struct** sockaddr \*src\_from, socklen\_t \*src\_len);

buffer用于储存接收到的数据，len指定buffer的长度，而flags在应用中通常被设置0，src\_from若不为空，则记录数据来源程序的IP端口，若src\_len不为空，则其长度信息记录在src\_len所指向的变量中。

注意：默认情况下，recvfrom是一个阻塞的调用，即直到它接收到数据才会返回。

5、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。

四、进程使用数据报socket进行通信

下面用多个客户程序实例和一个服务器程序来演示多个进程如何通过使用数据报socket来进行通信。

sockserver2.c是一个服务器程序，它接收客户程序发来的数据，并创建一个子进程来处理客户发送过来的数据，处理过程非常简单，就是把大写字母改为小写。然后把处理后的数据（大写字母对应的小写字母）发送回给客户端。

sockclient2.c是一个客户程序，它向服务器程序发送数据，并接收服务器发送过来的处理后的数据（即小写字母），然后把接收到的数据输出到屏幕上。在运行客户程序时，你可以为它提供一个字符作为参数，此时客户程序将把些字符作为要处理的数据发送给服务器，如果不提供一个参数，则默认发送字符A。

sockserver2.c的源代码如下：

**[cpp]** [view plain](http://blog.csdn.net/ljianhui/article/details/10697935) [copy](http://blog.csdn.net/ljianhui/article/details/10697935)

[print?](http://blog.csdn.net/ljianhui/article/details/10697935)

1. #include <unistd.h>
2. #include <sys/types.h>
3. #include <sys/socket.h>
4. #include <netinet/in.h>
5. #include <stdio.h>
6. #include <stdlib.h>
7. #include <signal.h>

10. **int** main()
11. {
12. **int** server\_sockfd = -1;
13. **int** server\_len = 0;
14. **int** client\_len = 0;
15. **char** buffer[512];
16. **int** result = 0;
17. **struct** sockaddr\_in server\_addr;
18. **struct** sockaddr\_in client\_addr;
19. //创建数据报套接字
20. server\_sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);
21. //设置监听IP端口
22. server\_addr.sin\_family = AF\_INET;
23. server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);
24. server\_addr.sin\_port = htons(9739);
25. server\_len = **sizeof**(server\_addr);
26. //绑定（命名）套接字
27. bind(server\_sockfd, (**struct** sockaddr\*)&server\_addr, server\_len);
28. //忽略子进程停止或退出信号
29. signal(SIGCHLD, SIG\_IGN);

32. **while**(1)
33. {
34. //接收数据，用client\_addr来储存数据来源程序的IP端口
35. result = recvfrom(server\_sockfd, buffer, **sizeof**(buffer), 0,
36. (**struct** sockaddr\*)&client\_addr, &client\_len);
37. **if**(fork() == 0)
38. {
39. //利用子进程来处理数据
40. buffer[0] += 'a' - 'A';
41. sleep(5);
42. //发送处理后的数据
43. sendto(server\_sockfd, buffer, **sizeof**(buffer),0 ,
44. (**struct** sockaddr\*)&client\_addr, client\_len);
45. printf("%c\n", buffer[0]);
46. //注意，一定要关闭子进程，否则程序运行会不正常
47. exit(0);
48. }
49. }
50. //关闭套接字
51. close(server\_sockfd);
52. }

sockclient2.c的源代码如下：

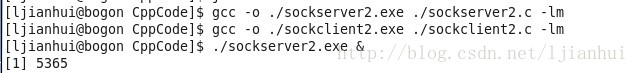
**[cpp]** [view plain](http://blog.csdn.net/ljianhui/article/details/10697935) [copy](http://blog.csdn.net/ljianhui/article/details/10697935)

[print?](http://blog.csdn.net/ljianhui/article/details/10697935)

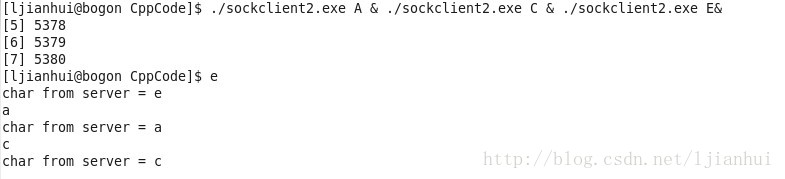
1. #include <unistd.h>
2. #include <sys/types.h>
3. #include <sys/socket.h>
4. #include <netinet/in.h>
5. #include <stdlib.h>
6. #include <stdio.h>
8. **int** main(**int** agrc, **char** \*argv[])
9. {
10. **struct** sockaddr\_in server\_addr;
11. **int** server\_len = 0;
12. **int** sockfd = -1;
13. **int** result = 0;
14. **char** c = 'A';
15. //取第一个参数的第一个字符
16. **if**(agrc > 1)
17. c = argv[1][0];
18. //创建数据报套接字
19. sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);
20. //设置服务器IP端口
21. server\_addr.sin\_family = AF\_INET;
22. server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");
23. server\_addr.sin\_port = htons(9739);
24. server\_len = **sizeof**(server\_addr);
25. //向服务器发送数据
26. sendto(sockfd, &c, **sizeof**(**char**), 0,
27. (**struct** sockaddr\*)&server\_addr, server\_len);
28. //接收服务器处理后发送过来的数据，由于不关心数据来源，所以把后两个参数设为0
29. recvfrom(sockfd, &c, **sizeof**(**char**), 0, 0, 0);
30. printf("char from server = %c\n", c);
31. //关闭套接字
32. close(sockfd);
33. exit(0);
34. }

运行结果如下：

先运行服务器程序，如下：



再运行三个客户端：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，即把大写字母变成了小写。recvfrom调用是阻塞的调用，即只有当接收到数据才会返回。

五、数据报套接字与流套接字的比较

1、从使用的便利和效率来讲

我们可以看到使用数据报套接字的确是比使用流套接字简单，而且快速。

因为使用流套接字的程序，客户程序需要调用connect来创建一个到服务器程序的连接，并需要维持这个连接，服务器程序也需要调用listen来创建一个队列来保存未处理的请求，当有数据到达时，服务器也不需要调用accept来接受连接并创建一个新socket描述符来处理请求。

再来看看使用数据报套接字的程序，服务器程序与客户程序所使用的系统调用大致相同，服务器程序只比客户程序多使用了一个bind调用。基于数据报套接字的程序，只需要使用sendto调用来向指定IP端口的程序发送信息，使用recvfrom调用从指向的IP端口接收信息即可。因为它并不需要建立一个连接，接受连接等，所以省去了很多的功夫。

2、从使用场合来讲

我们知道流套接字是基于TCP/IP协议的，它是一种安全的协议，提供的是一个有序、可靠、双向字节流的连接，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。所以它比较适合用来发送信息量大的数据文件，或对数据完整性要求较高的文件，如压缩文件、视频文件等

而数据报套接字是基于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。所以它比较适合发送一些对实时性要求较高，但是对安全性和完整性要求不太高的数据。如我们熟悉的聊天信息，即使有一点的丢失也不会造成理解上的大的问题。