## 第13课 讲稿（Linux网络编程）

1. **网络高级编程**

在实际情况中，人们往往遇到多个客户端连接服务器端的情况。由于之前介绍的如connet()、recv()和send()等都是阻塞性函数，如果资源没有准备好，则调用该函数的进程将进入睡眠状态，这样就无法处理I/O 多路复用的情况了。本节使用select()解决多路复用的问题。可以看到，由于在Linux 中把socket 也作为一种特殊文件描述符，这给用户的处理带来了很大的方便。

1. select()

下面是使用select()函数的服务器端源代码。客户端程序与前面的例子基本相同，仅在“close(sockfd)”前面加一句“sleep(5)”，令客户端延迟5秒后再退出。

|  |
| --- |
| **/\* net\_select.c \*/**  #include <sys/types.h>  #include <sys/socket.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/time.h>  #include <sys/ioctl.h>  #include <unistd.h>  #include <netinet/in.h>  #define PORT 4321  #define MAX\_QUE\_CONN\_NM 5  #define MAX\_SOCK\_FD FD\_SETSIZE  #define BUFFER\_SIZE 1024  int main()  {  struct sockaddr\_in server\_sockaddr, client\_sockaddr;  int sin\_size, count;  fd\_set inset, tmp\_inset;  int sockfd, client\_fd, fd;  char buf[BUFFER\_SIZE];  if ((sockfd = **socket(AF\_INET, SOCK\_STREAM, 0)**) == -1)  {  perror("socket");  exit(1);  }  **server\_sockaddr.sin\_family = AF\_INET;**  **server\_sockaddr.sin\_port = htons(PORT);**  **server\_sockaddr.sin\_addr.s\_addr = INADDR\_ANY;**  **bzero(&(server\_sockaddr.sin\_zero), 8);**  **int i = 1;**/\* 允许重复使用本地地址与套接字进行绑定 \*/  **setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &i, sizeof(i));**  if **(bind(sockfd, (struct sockaddr \*)&server\_sockaddr,**  **sizeof(struct sockaddr)**) == -1)  {  perror("bind");  exit(1);  }  if**(listen(sockfd, MAX\_QUE\_CONN\_NM)** == -1)  {  perror("listen");  exit(1);  }  printf("listening....\n");  /\*将调用socket()函数的描述符作为文件描述符\*/  **FD\_ZERO(&inset);**  **FD\_SET(sockfd, &inset);**  while(1)  {  tmp\_inset = inset;  sin\_size=sizeof(struct sockaddr\_in);  memset(buf, 0, sizeof(buf));  /\*调用select()函数\*/  if (!(**select(MAX\_SOCK\_FD, &tmp\_inset, NULL, NULL, NULL)** > 0))  {  perror("select");  }  for (fd = 0; fd < MAX\_SOCK\_FD; fd++)  {  if **(FD\_ISSET(fd, &tmp\_inset) > 0)**  {  if (fd == sockfd)  { /\* 服务端接收客户端的连接请求 \*/  if ((client\_fd = accept(sockfd,**(struct sockaddr \*)&client\_sockaddr, &sin\_size)**)== -1)  {  perror("accept");  exit(1);  }  **FD\_SET(client\_fd, &inset);**  printf("New connection from %d(socket)\n", client\_fd);  }  else /\* 处理从客户端发来的消息 \*/  {  if ((count = **recv(client\_fd, buf, BUFFER\_SIZE, 0)**) > 0)  {  printf("Received a message from %d: %s\n",client\_fd, buf);  }  else  {  **close(fd);**  **FD\_CLR(fd, &inset)**;  printf("Client %d(socket) has left\n", fd);  }  }  } /\* end of if FD\_ISSET\*/  } /\* end of for fd\*/  } /\* end if while while\*/  close(sockfd);  exit(0);  } |

运行该程序时，可以先启动服务器端，再反复运行客户端程序（这里启动两个客户端进程）即可，服务器端运行结果如下所示：

|  |
| --- |
| 在第一个终端中：  # ./net\_select  listening....  New connection from 4(socket) /\* 接受第一个客户端的连接请求\*/  Received a message from 4: **Hello,First!** /\* 接收第一个客户端发送的数据\*/  New connection from 5(socket) /\* 接受第二个客户端的连接请求\*/  Received a message from 5: **Hello,Second!** /\* 接收第二个客户端发送的数据\*/  Client 4(socket) has left /\* 检测到第一个客户端离线了\*/  Client 5(socket) has left /\* 检测到第二个客户端离线了\*/  在第二个终端中：  # .**/client localhost Hello,First! & ./client localhost Hello,Second** |