一、I/O 复用概述

I/O 复用概念：

解决进程或线程阻塞到某个 I/O 系统调用而出现的技术，使进程不阻塞于某个特定的 I/O 系统调

I/O 复用使用的场合：

1.当客户处理多个描述符（通常是交互式输入、网络套接字）时，必须使用 I/O 复用。

2.tcp 服务器既要处理监听套接字，又要处理已连接套接字，一般要使用 I/O 复用。

3.如果一个服务器既要处理 tcp 又要处理 udp，一般要使用 I/O 复用。

4.如果一个服务器要处理多个服务或多个服务时，一般要使用 I/O 复用。

I/O 复用常用函数：

select、poll

二、select()函数

select 函数介绍：

```
int select(int  maxfd, fd_set *readset,  fd_set
*writeset, fd_set *exceptset,  const struct timeval
*timeout);
```

功能：轮询扫描多个描述符中的任一描述符是否发生响应，或经过一段时间后唤醒

参数：

参数	名称	说明
maxfd	指定要检测的描述符的范围	所检测描述符最大值+1
readset	可读描述符集	监测该集合中的任意描述符是否有数据可读
writeset	可写描述符集	监测该集合中的任意描述符是否有数据可写
exceptset	异常描述符集	监测该集合中的任意描述符是否发生异常
timeout	超时时间	超过规定时间后唤醒

返回值：

0：超时

-1：出错

>0：准备好的文件描述符数量

头文件：

[csharp] view plain copy

```
#include <sys/select.h>
#include <sys/time.h>
```

超时时间：

[csharp] view plain copy

```
//该结构体表示等待超时的时间
struct timeval{
    long tv_sec;//秒
    long tv_usec;//微秒
};

//比如等待 10.2 秒
struct timeval timeout;
timeoout.tv_sec = 10;
timeoout.tv_usec = 200000;

//将 select 函数的 timeout 参数设置为 NULL 则永远等待
```

描述符集合的操作：

select()函数能对多个文件描述符进行监测，如果一个参数对应一个描述符，那么 select 函数的 4 个参数最多能监测 4 个文件描述，那他如何实现对多个文件描述符的监测的呢？

大家想一想文件描述符基本具有 3 种特性（读、写、异常），如果我们统一将监测可读的描述符放入可读集合（readset），监测可写的描述符放入可写集合（writeset），监测异常的描述符放入异常集合（exceptset）。然后将这 3 个集合传给 select 函数，是不是就可监测多个描述符呢.

如何将某个描述符加入到特定的集合中呢？这时就需要了解下面的集合操作函数

[csharp] view plain copy

```
1. //初始化描述符集
2. void FD_ZERO(fd_set *fdset);
3.
4. //将一个描述符添加到描述符集
5. void FD_SET(int fd, fd_set *fdset);
6.
7. //将一个描述符从描述符集中删除
8. void FD_CLR(int fd, fd_set *fdset);
9.
10. //检测指定的描述符是否有事件发生
11. int FD_ISSET(int fd, fd_set *fdset);
```

select()函数整体使用框架：

例子：检测 0、4、5 描述符是否准备好读

[csharp] view plain copy

```
1. while(1)
2. {
3.     fd_set rset;//创建一个描述符集 rset
4.     FD_ZERO(&rset);//对描述符集 rset 清零
5.     FD_SET(0, &rset);//将描述符 0 加入到描述符集 rset 中
```

```
    FD_SET(4, &rset);//将描述符 4 加入到描述符集 rset 中
    FD_SET(5, &rset);//将描述符 5 加入到描述符集 rset 中

    if(select(5+1, &rset, NULL, NULL, NULL) > 0)
     {
         if(FD_ISSET(0, &rset))
         {
             //描述符 0 可读及相应的处理代码
         }

         if(FD_ISSET(4, &rset))
         {
             //描述符 4 可读及相应的处理代码
         }
         if(FD_ISSET(5, &rset))
         {
             //描述符 5 可读及相应的处理代码
         }
     }
}
```

三、select 函数的应用对比

我们通过 udp 同时收发的例子来说明 select 的妙处。

对于 udp 同时收发立马想到的是一个线程收、另一个线程发，下面的代码就是通过多线程来实现

[csharp] view plain copy

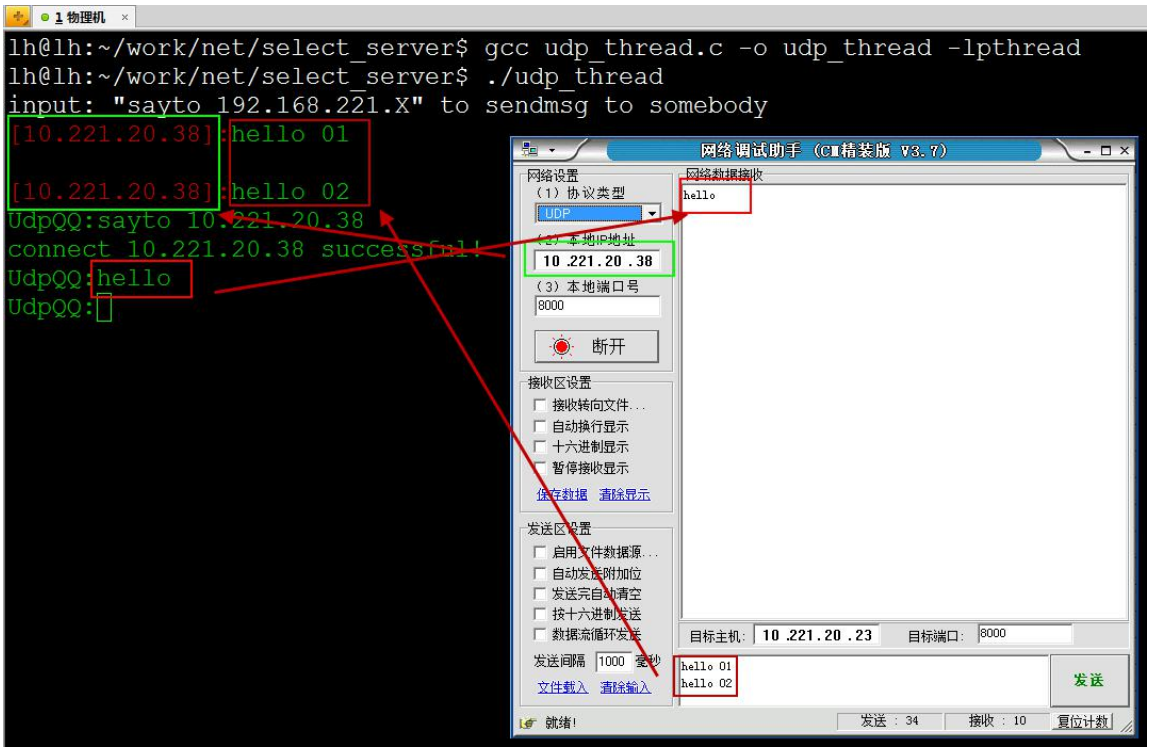
```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/select.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <arpa/inet.h>
#include <pthread.h>

//接收线程：负责接收消息并显示
void *recv_thread(void* arg)
{
    int udpfd = (int)arg;
    struct sockaddr_in addr;
    socklen_t addrlen = sizeof(addr);

    bzero(&addr,sizeof(addr));
    while(1)
    {
        char buf[200]  = "";
        char ipbuf[16] = "";
        recvfrom(udpfd, buf, sizeof(buf), 0, (struct sockaddr*)&addr, &addrlen);
        printf("\r\033[31m[%s]:\033[32m%s\n",inet_ntop(AF_INET,&addr.sin_addr,ipbuf,sizeof(ipbuf)),buf);
        write(1,"UdpQQ:",6);
    }
    return NULL;
}

int main(int argc,char *argv[])
{
    char buf[100] = "";
    int  udpfd = 0;
    pthread_t tid;
    struct sockaddr_in addr;
    struct sockaddr_in cliaddr;

    //对套接字地址进行初始化
    bzero(&addr,sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port   = htons(8000);
    addr.sin_addr.s_addr = htonl(INADDR_ANY);

    bzero(&cliaddr,sizeof(cliaddr));
    cliaddr.sin_family = AF_INET;
    cliaddr.sin_port   = htons(8000);

    //创建套接口
    if( (udpfd = socket(AF_INET,SOCK_DGRAM, 0)) < 0)
```

```
51.     {
52.         perror("socket error");
53.         exit(-1);
54.     }
55.
56.     //设置端口
57.     if(bind(udpfd, (struct sockaddr*)&addr, sizeof(addr)) < 0)

58.     {
59.         perror("bind error");
60.         close(udpfd);
61.         exit(-1);
62.     }
63.
64.     printf("input: \"sayto 192.168.221.X\" to sendmsg to someb
    ody\n");
65.     //创建接收线程
66.     pthread_create(&tid, NULL, recv_thread, (void*)udpfd); //
    创建线程
67.     printf("\033[32m"); //设置字体颜色
68.     fflush(stdout);
69.
70.     while(1)
71.     {
72.         //主线程负责发送消息
73.         write(1,"UdpQQ:",6);//1 表示标准输出
74.         fgets(buf, sizeof(buf), stdin); //等待输入
75.         buf[strlen(buf) - 1] = '\0';    //确保输入的最后一位是
    '\0'
76.         if(strncmp(buf, "sayto", 5) == 0)
77.         {
78.             char ipbuf[INET_ADDRSTRLEN] = "";
79.             inet_pton(AF_INET, buf+6, &cliaddr.sin_addr);//给
    addr 套接字地址再赋值.
80.             printf("\rconnect %s successful!\n",inet_ntop(AF_I
    NET,&cliaddr.sin_addr,ipbuf,sizeof(ipbuf)));
81.             continue;
82.         }
83.         else if(strncmp(buf, "exit",4) == 0)
84.         {
85.             close(udpfd);
86.             exit(0);
87.         }
88.
```

```
89.         sendto(udpfd, buf, strlen(buf),0,(struct sockaddr*)&cl
    iaddr, sizeof(cliaddr));
90.     }
91.
92.     return 0;
93. }
```

运行结果：



用 select 来完成上述同样的功能：

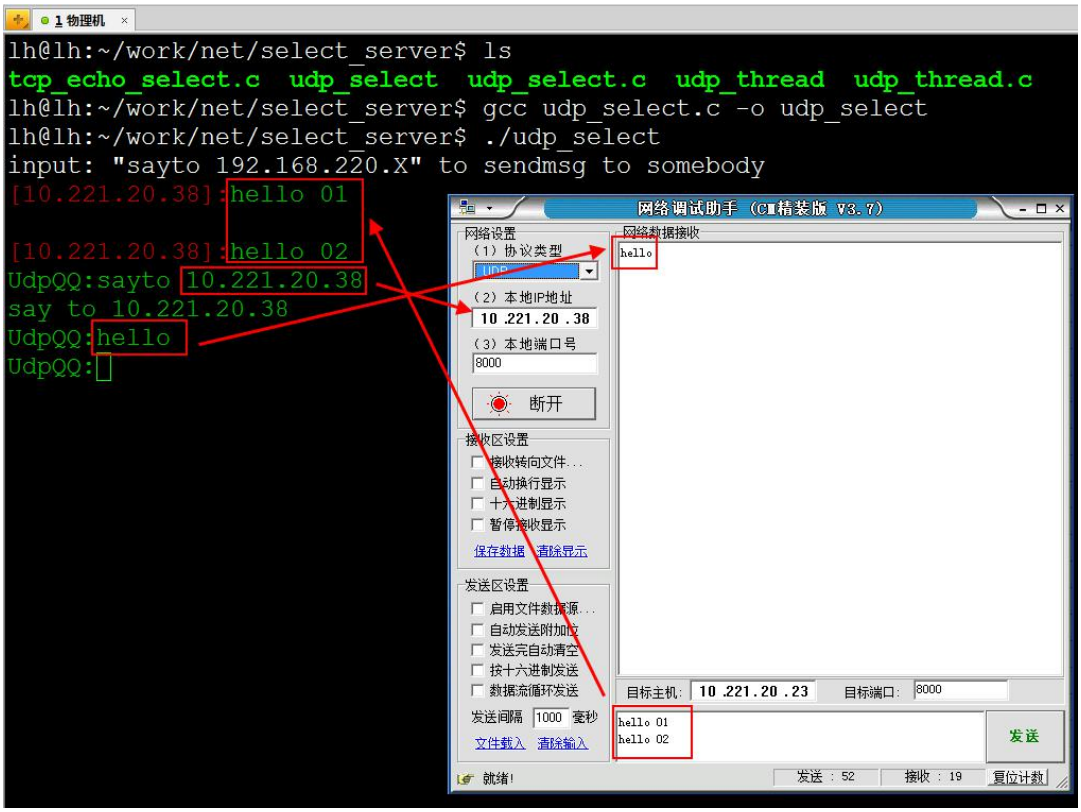
[csharp] view plain copy

```
1. #include <string.h>
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <unistd.h>
5. #include <sys/select.h>
6. #include <sys/time.h>
7. #include <sys/socket.h>
8. #include <netinet/in.h>
```

```c
#include <arpa/inet.h>

int main(int argc,char *argv[])
{
    int udpfd = 0;
    struct sockaddr_in saddr;
    struct sockaddr_in caddr;

    bzero(&saddr,sizeof(saddr));
    saddr.sin_family = AF_INET;
    saddr.sin_port   = htons(8000);
    saddr.sin_addr.s_addr = htonl(INADDR_ANY);

    bzero(&caddr,sizeof(caddr));
    caddr.sin_family  = AF_INET;
    caddr.sin_port    = htons(8000);

    //创建套接字
    if( (udpfd = socket(AF_INET,SOCK_DGRAM, 0)) < 0)
    {
        perror("socket error");
        exit(-1);
    }

    //套接字端口绑字
    if(bind(udpfd, (struct sockaddr*)&saddr, sizeof(saddr)) != 0)
    {
        perror("bind error");
        close(udpfd);
        exit(-1);
    }

    printf("input: \"sayto 192.168.220.X\" to sendmsg to somebody\033[32m\n");
    while(1)
    {
        char buf[100]="";
        fd_set rset;    //创建文件描述符的聚合变量
        FD_ZERO(&rset); //文件描述符聚合变量清 0
        FD_SET(0, &rset);//将标准输入添加到文件描述符聚合变量中
        FD_SET(udpfd, &rset);//将 udpfd 添加到文件描述符聚合变量中

        write(1,"UdpQQ:",6);
```

```
50.
51.         if(select(udpfd + 1, &rset, NULL, NULL, NULL) > 0)
52.         {
53.             if(FD_ISSET(0, &rset))//测试 0 是否可读写
54.             {
55.                 fgets(buf, sizeof(buf), stdin);
56.                 buf[strlen(buf) - 1] = '\0';
57.                 if(strncmp(buf, "sayto", 5) == 0)
58.                 {
59.                     char ipbuf[16] = "";
60.                     inet_pton(AF_INET, buf+6, &caddr.sin_addr);
    //给 addr 套接字地址再赋值.
61.                     printf("\rsay to %s\n",inet_ntop(AF_INET,&
    caddr.sin_addr,ipbuf,sizeof(ipbuf)));
62.                     continue;
63.                 }
64.                 else if(strcmp(buf, "exit")==0)
65.                 {
66.                     close(udpfd);
67.                     exit(0);
68.                 }
69.                 sendto(udpfd, buf, strlen(buf),0,(struct socka
    ddr*)&caddr, sizeof(caddr));
70.             }
71.             if(FD_ISSET(udpfd, &rset))//测试 udpfd 是否可读写
72.             {
73.                 struct sockaddr_in addr;
74.                 char ipbuf[INET_ADDRSTRLEN] = "";
75.                 socklen_t addrlen = sizeof(addr);
76.
77.                 bzero(&addr,sizeof(addr));
78.
79.                 recvfrom(udpfd, buf, 100, 0, (struct sockaddr*)
    &addr, &addrlen);
80.                 printf("\r\033[31m[%s]:\033[32m%s\n",inet_ntop
    (AF_INET,&addr.sin_addr,ipbuf,sizeof(ipbuf)),buf);
81.             }
82.         }
83.     }
84.
85.     return 0;
86. }
```

运行结果：


1 物理机
lh@lh:~/work/net/select_server$ ls
tcp_echo_select.c udp_select udp_select.c udp_thread udp_thread.c
lh@lh:~/work/net/select_server$ gcc udp_select.c -o udp_select
lh@lh:~/work/net/select_server$ ./udp_select
input: "sayto 192.168.220.X" to sendmsg to somebody
[10.221.20.38]:hello 01
[10.221.20.38]:hello 02
UdpQQ:sayto 10.221.20.38
say to 10.221.20.38
UdpQQ:hello
UdpQQ:
网络调试助手 (C1精装版 V3.7)
网络设置
(1) 协议类型
(2) 本地IP地址
10 .221.20 .38
(3) 本地端口号
8000
断开
接收区设置
接收转向文件...
自动换行显示
十六进制显示
暂停接收显示
保存数据 清除显示
发送区设置
启用文件数据源...
自动发送附加位
发送完自动清空
按十六进制发送
数据流循环发送
发送间隔 1000 毫秒
文件载入 清除输入
网络数据接收
hello
目标主机: 10 .221.20 .23
目标端口: 8000
hello 01
hello 02
发送
就绪!
发送 : 52
接收 : 19
复位计数

代码下载：