知识点1【select的函数的概述】（了解）

1、select函数概述

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

int select(int nfds, fd_set *readfds, fd_set *writefds,
fd_set *exceptfds, struct timeval *timeout);
```

select允许程序可以监视多个文件描述符（套接字）的（可读，可写，异常），直到一个或多个文件描述符（套接字）准备就绪（超时），select解阻塞继续执行。

2、select参数介绍

nfds:

是这三个集合中编号最高的文件描述符是加1。

readfds:

监视readfds集合中的描述符是否读操作准备就绪。

writefds:

监视writefds集合中的描述符是否写操作准备就绪

exceptfds:

监视exceptfds集合中的描述符是否发生了异常。

timeout:

超时时间，如果在超时时间内没有任意描述符准备好或异常，那么select超时时间一到也会解阻塞

```
struct timeval
{
 long    tv_ sec;         /* seconds */
 long    tv_ usec;      /* microseconds */
};
```

timeout有三种情况:

timeout为NULL 表示永远等待下去

timeout中所有成员问0， 不等待立即解阻塞

timeout中成员有固定的时间 表示等待该时间

返回值:

> 0：表示就绪的文件描述符数量。

=0：表示超时。

-1：表示select出错。

3、readfds、writefds、exceptfds都是fd_set文件描述符集合

所以必须学会文件描述符集合的操作

知识点2【描述符集合操作API】（了解）

1、集合类型

```
fd_set
```

2、清空集合

```
void FD_ ZERO( fd_ set *fdset);
```

3、将fd描述符 添加到集合中

```
void FD_ SET( int fd, fd_ set *fdset);
```

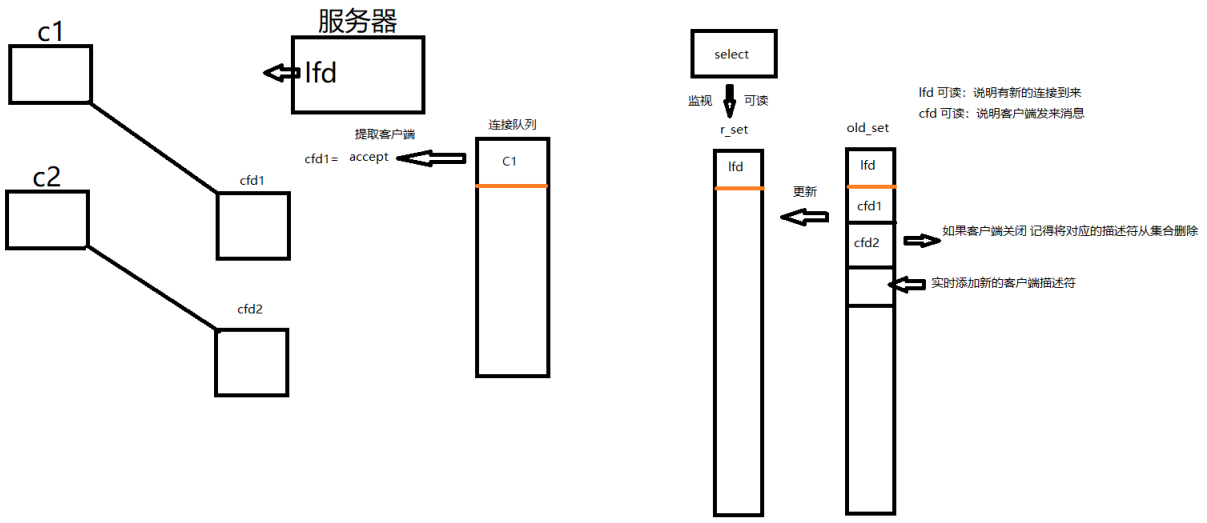
4、将fd描述符 从集合中删除

```
void FD_CLR( int fd, fd_ set *fdset);
```

5、判断fd描述符 是否在集合中

```
int FD_ ISSET( int fd, fd_ set *fdset);
```

知识点3【基于TCP的select并发echo服务器】（了解）




```
#include <stdio.h>
#include <sys/socket.h> //socket
#include <unistd.h>     //_exit
#include <netinet/in.h> //struct sockaddr_in
#include <string.h>     //bzero
#include <sys/time.h>
```

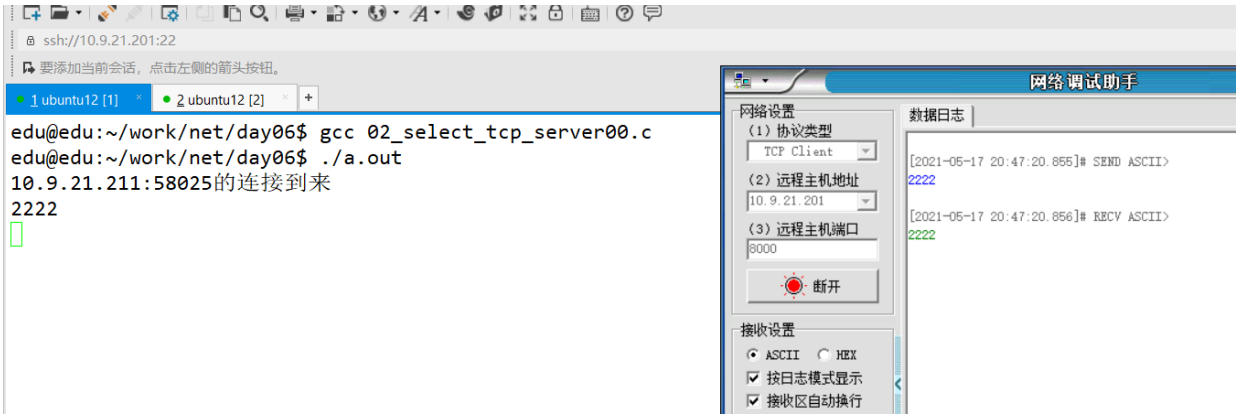
```c
#include <sys/types.h> //select
#include <arpa/inet.h> //inet_ntop
int create_tcp_socket(unsigned short port)
{
    //创建tcp监听套接字
    int lfd = socket(AF_INET, SOCK_STREAM, 0);
    if (lfd < 0)
    {
        perror("socket");
        _exit(-1);
    }

    //设置端口复用
    int yes = 1;
    setsockopt(lfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes));

    //bind给lfd绑定固定的ip port
    struct sockaddr_in my_addr;
    bzero(&my_addr, sizeof(my_addr));
    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(port);
    my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    int ret = bind(lfd, (struct sockaddr *)&my_addr, sizeof(my_addr));
    if (ret < 0)
    {
        perror("bind");
        _exit(-1);
    }

    //listen进行监听
    listen(lfd, 128);

    return lfd;
}
int main(int argc, char const *argv[])
{
    int lfd = create_tcp_socket(8000);

    //将lfd添加到old_set集合中
    fd_set old_set;
```

```
47      //old_set清空
48      FD_ZERO(&old_set);
49      //将lfd放入old_set集合中
50      FD_SET(lfd, &old_set);
51      int max_fd = lfd;
52
53      //select需要循环的监听所有文件描述符的状态
54      while (1)
55      {
56          fd_set r_set = old_set;
57          //表示只对r_set集合监听读操作
58          int nready = select(max_fd + 1, &r_set, NULL, NULL, NULL);
59          if (nready < 0)
60          {
61              perror("select");
62              break;
63          }
64          else if (nready > 0) //有多个文件描述符准备好了
65          {
66              //由于lfd和cfd功能不一样 所以需要单独处理lfd
67              if (FD_ISSET(lfd, &r_set)) //判断lfd如果在r_set 说明lfd可读
（有新的连接到来）
68              {
69                  //提取与客户端通信的已连接套接字cfd
70                  struct sockaddr_in cli_addr;
71                  socklen_t cli_len = sizeof(cli_addr);
72                  int cfd = accept(lfd, (struct sockaddr *)&cli_addr, &cli
_len);
73
74                  //打印一下客户端的信息
75                  char ip[16] = "";
76                  printf("%s:%hu的连接到来\n",
77                         inet_ntop(AF_INET, &cli_addr.sin_addr.s_addr, ip,
6),
78                         ntohs(cli_addr.sin_port));
79
80                  //将cfd放入old_set集合中
81                  FD_SET(cfd, &old_set);
82
83                  //更新最大值  max_fd
84                  if (max_fd < cfd)
```

```
85                      max_fd = cfd;
86
87                  //nready ==0说明当前只有lfd准备就绪
88                  if (--nready == 0)
89                      continue;
90              }
91
92              //已连接套接字cfd 准备就绪
93              int i = 0;
94              //逐个cfd判断是否在r_set集合（准备就绪）中
95              for (i = lfd + 1; i <= max_fd; i++)
96              {
97                  if (FD_ISSET(i, &r_set)) //i文件描述符准备就绪
98                  {
99                      //recv读取客户端的信息
100                     unsigned char buf[1500] = "";
101                     int n = recv(i, buf, sizeof(buf), 0);
102                     if (n < 0) //错误
103                     {
104                         perror("recv");
105                     }
106                     else if (n == 0) //对方关闭
107                     {
108                         //将i从old_set集合中删除
109                         FD_CLR(i, &old_set);
110                         //关闭套接字
111                         close(i);
112                         printf("已有客户端退出\n");
113                     }
114                     else if (n > 0) //收到客户端的信息
115                     {
116                         //将接收到的buf原样的发送给客户端
117                         send(i, buf, n, 0);
118                         printf("%s\n", buf);
119                     }
120                 }
121             }
122         }
123     }
124
```

```
125      close(lfd);
126      return 0;
127  }
```

运行结果：



知识点4【基于TCP的select并发echo服务器】（升级）

查看进程打开的文件描述符

/proc/进程号/fd中

```
1  #include <stdio.h>
2  #include <sys/socket.h> //socket
3  #include <unistd.h>     //_exit
4  #include <netinet/in.h> //struct sockaddr_in
5  #include <string.h>     //bzero
6  #include <sys/time.h>
7  #include <sys/types.h> //select
8  #include <arpa/inet.h> //inet_ntop
9  int create_tcp_socket(unsigned short port)
10 {
11     //创建tcp监听套接字
12     int lfd = socket(AF_INET, SOCK_STREAM, 0);
13     if (lfd < 0)
14     {
15         perror("socket");
16         _exit(-1);
17     }
18
19     //设置端口复用
20     int yes = 1;
```

```
    setsockopt(lfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes));

    //bind给lfd绑定固定的ip port
    struct sockaddr_in my_addr;
    bzero(&my_addr, sizeof(my_addr));
    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(port);
    my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    int ret = bind(lfd, (struct sockaddr *)&my_addr, sizeof(my_addr));
    if (ret < 0)
    {
        perror("bind");
        _exit(-1);
    }

    //listen进行监听
    listen(lfd, 128);

    return lfd;
}
int main(int argc, char const *argv[])
{
    int lfd = create_tcp_socket(8000);

    //将lfd添加到old_set集合中
    fd_set old_set;
    //old_set清空
    FD_ZERO(&old_set);
    //将lfd放入old_set集合中
    FD_SET(lfd, &old_set);
    int max_fd = lfd;

    //select需要循环的监听所有文件描述符的状态
    while (1)
    {
        fd_set r_set = old_set;
        //表示只对r_set集合监听读操作
        int nready = select(max_fd + 1, &r_set, NULL, NULL, NULL);
        if (nready < 0)
        {
```

```
61              perror("select");
62              break;
63          }
64          else if (nready > 0) //有多个文件描述符准备好了
65          {
66              //由于lfd和cfd功能不一样 所以需要单独处理lfd
67              if (FD_ISSET(lfd, &r_set)) //判断lfd如果在r_set 说明lfd可读（有新的连接到来）
68              {
69                  //提取与客户端通信的已连接套接字cfd
70                  struct sockaddr_in cli_addr;
71                  socklen_t cli_len = sizeof(cli_addr);
72                  int cfd = accept(lfd, (struct sockaddr *)&cli_addr, &cli_len);
73
74                  //打印一下客户端的信息
75                  char ip[16] = "";
76                  printf("%s:%hu的连接到来\n",
77                         inet_ntop(AF_INET, &cli_addr.sin_addr.s_addr, ip, 6),
78                         ntohs(cli_addr.sin_port));
79
80                  //将cfd放入old_set集合中
81                  FD_SET(cfd, &old_set);
82
83                  //更新最大值  max_fd
84                  if (max_fd < cfd)
85                      max_fd = cfd;
86
87                  //nready ==0说明当前只有lfd准备就绪
88                  if (--nready == 0)
89                      continue;
90              }
91
92              //已连接套接字cfd 准备就绪
93              int i = 0;
94              //逐个cfd判断是否在r_set集合（准备就绪）中
95              for (i = lfd + 1; i <= max_fd; i++)
96              {
97                  if (FD_ISSET(i, &r_set)) //i文件描述符准备就绪
98                  {
```

```
                    //recv读取客户端的信息
                    unsigned char buf[1500] = "";
                    int n = recv(i, buf, sizeof(buf), 0);
                    if (n < 0) //错误
                    {
                        perror("recv");
                    }
                    else if (n == 0) //对方关闭
                    {
                        //最大文件描述符将关闭
                        if (i == max_fd)
                        {
                            //将i从old_set集合中删除
                            FD_CLR(i, &old_set);
                            //关闭套接字
                            close(i);
                            max_fd--;
                        }
                        else
                        {
                            //先关闭i  i也指向max_fd
                            dup2(max_fd, i);
                            //载关闭max_fd 并更新max_fd
                            close(max_fd);
                            //将max_fd从old_set集合中删除
                            FD_CLR(max_fd, &old_set);
                            max_fd--;
                        }
                        printf("已有客户端退出max_fd=%d\n", max_fd);
                    }
                    else if (n > 0) //收到客户端的信息
                    {
                        //将接收到的buf原样的发送给客户端
                        send(i, buf, n, 0);
                        printf("%s\n", buf);
                    }
                }
            }
        }
    }
```

```
139
140     close(lfd);
141     return 0;
142 }
```

知识点5【select优缺点】

优点：

跨平台windows、linux都可以使用

缺点：

有1024文件描述符的限制 FD_SETSIZE

每次重新监听都需要将监听的文件描述符集合 从用户态拷贝至 内核态

大量并发、少量活跃，效率低 （无解）