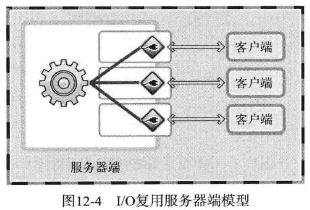
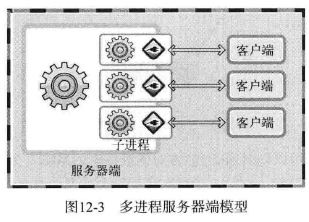
1. IO复用模型与多进程模型对比：

a. 多进程模型：需要大量的运算和内存空间，进程间数据交换复杂IPC。

b. IO复用模型可以不创建多进程同时又向多个客户端提供服务，但是并不适用于所有情况。



2. select函数，将多个文件描述符集中在一起统一监视。

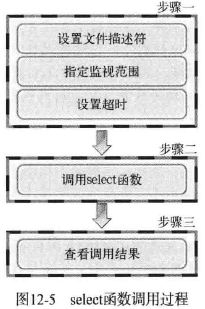
监视项目有：

是否存在套接字接收数据；

无需阻塞传输数据的套接字有哪些；

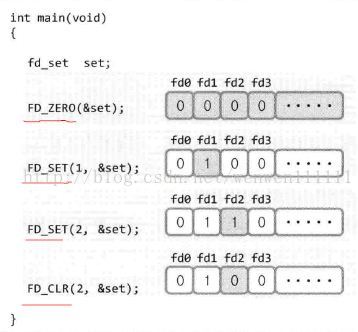
哪些套接字发生异常；

三个集合：{可读、可写和异常}



3. 设置文件描述符

1. FD\_ZERO(fd\_set \*fdset) //将fd\_set变量的所有位初始化0
2. FD\_SET(**int** fd, fd\_set \*fdset) //注册文件描述符fd到fdset （以位数组的形式注册）
3. FD\_CLR(**int** fd, fd\_set \*fdset) //清除文件描述符fd
4. FD\_ISSET(**int** fd, fd\_set \*fdset) //判断是否包含



1. 设置检查范围和超时
2. #include <sys/select.h>
3. #include <sys/time.h>
5. **int** select(**int** maxfd,fd\_set \*readset,fd\_set \*writeset,fd\_set \*exceptset, **const** **struct** timeval \* timeout)
7. 成功返回大于0的值，超时返回0，失败返回-1
9. maxfd：监视对象文件描述符数量 （最大文件描述符+1）
10. readset：是否存在待读取数据的文件描述符
11. writeset：是否可传输无阻塞数据的文件描述符
12. exceptset：是否发生异常的文件描述符
13. **timeout：超时信息 （如果select发生了阻塞，那么就通过设置timeout防止这种情况）**

监视范围：与第一个参数有关，最大的文件描述符值加1（加1,是因为文件描述符从0开始）。

超时：

1. **struct** timeval{
2. **long** tv\_sec; //sec
3. **long** tv\_usec; // microsec
4. }

5. IO复用的服务器端

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <unistd.h>
5. #include <arpa/inet.h>
6. #include <signal.h>
7. #include <sys/wait.h>
8. #include <sys/socket.h>
9. #include <sys/time.h>
10. #include <sys/select.h>
12. #define BUF\_SIZE    100
14. **void** error\_handling(**char** \*message);
16. **int** main(**int** argc, **char** \*argv[]){
17. **int** serv\_sock,clnt\_sock;
18. **char** buf[BUF\_SIZE];
19. **struct** sockaddr\_in serv\_adr;
20. **struct** sockaddr\_in clnt\_adr;
21. socklen\_t adr\_sz;
22. fd\_set reads,cpy\_reads;
23. **struct** timeval timeout;
24. **int** fd\_max,str\_len,fd\_num,i;
26. **if**(argc != 2){
27. printf("Usage : %s <port>\n",argv[0]);
28. exit(1);
29. }
31. serv\_sock = socket(PF\_INET,SOCK\_STREAM,0);
32. **if**(serv\_sock == -1){
33. error\_handling("socket error");
34. }
36. memset(&serv\_adr,0,**sizeof**(serv\_adr));
37. serv\_adr.sin\_family=AF\_INET;
38. serv\_adr.sin\_addr.s\_addr=htonl(INADDR\_ANY);
39. serv\_adr.sin\_port=htons(atoi(argv[1]));
41. **if**(bind(serv\_sock,(**struct** sockaddr\*)&serv\_adr,**sizeof**(serv\_adr)) == -1){
42. error\_handling("bind() error");
43. }
45. **if**(listen(serv\_sock,5) == -1){
46. error\_handling("listen() error");
47. }
49. FD\_ZERO(&reads);
50. FD\_SET(serv\_sock,&reads);
51. fd\_max = serv\_sock; //为什么要用serv\_sock作为最大文件描述符，这个地方没看懂
52. 尽管后面进行了求解最大值，但我认为这地方有点不妥///
53. **while**(1){
54. cpy\_reads = reads;  //因为进行复用的时候，需要对原有的文件描述符位数组进行修 改，为了不修改原数组，这里进行了拷贝
55. timeout.tv\_sec = 5;
56. timeout.tv\_usec = 5000;
58. **if**((fd\_num = select(fd\_max+1,&cpy\_reads,0,0,&timeout)) == -1)
59. **break**;
60. **if**(fd\_num == 0)
61. **continue**;
62. **for**(i=0;i<fd\_max+1;i++){
63. **if**(FD\_ISSET(i,&cpy\_reads)){
64. **if**(i == serv\_sock){ //connection request
65. adr\_sz = **sizeof**(clnt\_adr);
66. clnt\_sock = accept(serv\_sock,(**struct** sockaddr \*)&clnt\_adr,&adr\_sz);
67. FD\_SET(clnt\_sock,&reads);
69. **if**(fd\_max < clnt\_sock){
70. fd\_max = clnt\_sock;  //求文件描述符的最大值
71. }
72. printf("connected client : %d \n", clnt\_sock);
73. }**else**{  //read message
74. str\_len = read(i,buf,BUF\_SIZE);
76. **if**(str\_len == 0){   //close
77. FD\_CLR(i,&reads);
78. close(i);
79. printf("closed client %d \n",i);
80. }**else**{
81. write(i,buf,str\_len);   //echo
82. }
83. }
84. }
85. }
86. }
87. close(serv\_sock);
89. **return** 0;
90. }
92. **void** error\_handling(**char** \*message){
94. fputs(message,stderr);
95. fputs("\n",stderr);
96. exit(1);
97. }