<https://blog.csdn.net/turkeyzhou/article/details/8609360>

**函数作用：**

系统提供select函数来实现多路复用输入/输出模型。select系统调用是用来让我们的程序监视多个文件句柄的状态变化的。程序会停在select这里等待，直到被监视的文件句柄有一个或多个发生了状态改变。关于文件句柄，其实就是一个整数，我们最熟悉的句柄是0、1、2三个，0是标准输入，1是标准输出，2是标准错误输出。0、1、2是整数表示的，对应的FILE \*结构的表示就是stdin、stdout、stderr。

函数原型：

**[cpp]** [view plaincopy](http://blog.csdn.net/tianmo2010/article/details/6595998)

1. **int** select(**int** maxfd,fd\_set \*rdset,fd\_set \*wrset, \
2. fd\_set \*exset,**struct** timeval \*timeout);

**参数说明：**

参数maxfd是需要监视的最大的文件描述符值+1；rdset,wrset,exset分别对应于需要检测的可读文件描述符的集合，可写文件描述符的集 合及异常文件描述符的集合。struct timeval结构用于描述一段时间长度，如果在这个时间内，需要监视的描述符没有事件发生则函数返回，返回值为0。

下面的宏提供了处理这三种描述词组的方式:  
FD\_CLR(inr fd,fd\_set\* set)；用来清除描述词组set中相关fd 的位  
FD\_ISSET(int fd,fd\_set \*set)；用来测试描述词组set中相关fd 的位是否为真  
FD\_SET（int fd,fd\_set\*set）；用来设置描述词组set中相关fd的位  
FD\_ZERO（fd\_set \*set）；用来清除描述词组set的全部位

参数timeout为结构timeval，用来设置select()的等待时间，其结构定义如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/tianmo2010/article/details/6595998)

1. **struct** timeval
2. {
3. **time\_t** tv\_sec;//second
4. **time\_t** tv\_usec;//minisecond
5. };

如果参数timeout设为：

NULL，则表示select（）没有timeout，select将一直被阻塞，直到某个文件描述符上发生了事件。

0：仅检测描述符集合的状态，然后立即返回，并不等待外部事件的发生。

特定的时间值：如果在指定的时间段里没有事件发生，select将超时返回。

**函数返回值：**

执行成功则返回文件描述词状态已改变的个数，如果返回0代表在描述词状态改变前已超过timeout时间，没有返回；当有错误发生时则返回-1，错误原因存于errno，此时参数readfds，writefds，exceptfds和timeout的值变成不可预测。错误值可能为：  
EBADF 文件描述词为无效的或该文件已关闭  
EINTR 此调用被信号所中断  
EINVAL 参数n 为负值。  
ENOMEM 核心内存不足

常见的程序片段如下：

fs\_set readset；  
FD\_ZERO(&readset);  
FD\_SET(fd,&readset);  
select(fd+1,&readset,NULL,NULL,NULL);  
if(FD\_ISSET(fd,readset){……}

**理解select模型：**

理解select模型的关键在于理解fd\_set,为说明方便，取fd\_set长度为1字节，fd\_set中的每一bit可以对应一个文件描述符fd。则1字节长的fd\_set最大可以对应8个fd。

（1）执行fd\_set set; FD\_ZERO(&set);则set用位表示是0000,0000。

（2）若fd＝5,执行FD\_SET(fd,&set);后set变为0001,0000(第5位置为1)

（3）若再加入fd＝2，fd=1,则set变为0001,0011

（4）执行select(6,&set,0,0,0)阻塞等待

（5）若fd=1,fd=2上都发生可读事件，则select返回，此时set变为0000,0011。注意：没有事件发生的fd=5被清空。

　基于上面的讨论，可以轻松得出select模型的特点：

　　（1)可监控的文件描述符个数取决与sizeof(fd\_set)的值。我这边服务 器上sizeof(fd\_set)＝512，每bit表示一个文件描述符，则我服务器上支持的最大文件描述符是512\*8=4096。据说可调，另有说虽 然可调，但调整上限受于编译内核时的变量值。本人对调整fd\_set的大小不太感兴趣，参考[http://www.cppblog.com](http://home.eeworld.com.cn/my/link.php?url=http://www.cppblog.com%2F) /CppExplore/archive/2008/03/21/45061.html中的模型2（1）可以有效突破select可监控的文件描述符上 限。

　　（2）将fd加入select监控集的同时，还要再使用一个数据结构array保存放到select监控集中的fd，一是用于再select 返回后，array作为源数据和fd\_set进行FD\_ISSET判断。二是select返回后会把以前加入的但并无事件发生的fd清空，则每次开始 select前都要重新从array取得fd逐一加入（FD\_ZERO最先），扫描array的同时取得fd最大值maxfd，用于select的第一个 参数。

　　（3）可见select模型必须在select前循环array（加fd，取maxfd），select返回后循环array（FD\_ISSET判断是否有时间发生）。

下面给一个伪码说明基本select模型的服务器模型：

**[cpp]** [view plaincopy](http://blog.csdn.net/tianmo2010/article/details/6595998)

1. array[slect\_len];
2. nSock=0;
3. array[nSock++]=listen\_fd;(之前listen port已绑定并listen)
4. maxfd=listen\_fd;
6. **while**(1){
8. FD\_ZERO(&set);
10. foreach (fd in array)
11. {
12. fd大于maxfd，则maxfd=fd
13. FD\_SET(fd,&set)
14. }
16. res=select(maxfd+1,&set,0,0,0)；
18. **if**(FD\_ISSET(listen\_fd,&set))
19. {
20. newfd=accept(listen\_fd);
21. array[nsock++]=newfd;
22. **if**(--res<=0) **continue**;
23. }
25. foreach 下标1开始 (fd in array)
26. {
27. **if**(FD\_ISSET(fd,&tyle="COLOR: #ff0000">set))
28. 执行读等相关操作
29. 如果错误或者关闭，则要删除该fd，将array中相应位置和最后一个元素互换就好，nsock减一
30. **if**(--res<=0) **continue**;
31. }
33. }

检测键盘有无输入，完整的程序如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/tianmo2010/article/details/6595998)

1. #include<sys/time.h>
2. #include<sys/types.h>
3. #include<unistd.h>
4. #include<string.h>
5. #include<stdlib.h>
6. #include<stdio.h>
7. **int** main()
8. {
9. **char** buf[10]="";
10. fd\_set rdfds;
11. **struct** timeval tv;
12. **int** ret;
13. FD\_ZERO(&rdfds);
14. FD\_SET(0,&rdfds);   //文件描述符0表示stdin键盘输入
15. tv.tv\_sec = 3;
16. tv.tv\_usec = 500;
17. ret = select(1,&rdfds,NULL,NULL,&tv);
18. **if**(ret<0)
19. printf("\n selcet");
20. **else** **if**(ret == 0)
21. printf("\n timeout");
22. **else**
23. printf("\n ret = %d",ret);
25. **if**(FD\_ISSET(1,&rdfds))  //如果有输入，从stdin中获取输入字符
26. {
27. printf("\n reading");
28. fread(buf,9,1,stdin);
29. }
30. write(1,buf,strlen(buf));
31. printf("\n %d \n",strlen(buf));
32. **return** 0;
33. }
34. //执行结果ret = 1.

利用Select模型，设计的web服务器：

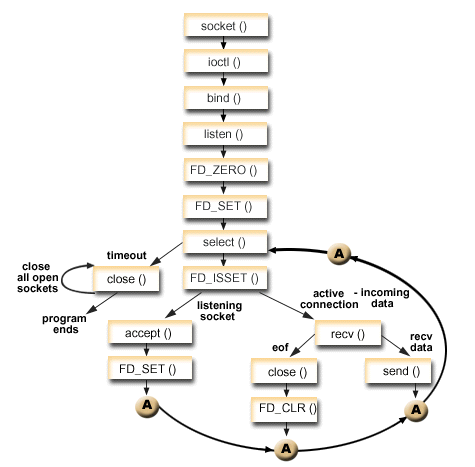
**[cpp]** [view plaincopy](http://blog.csdn.net/tianmo2010/article/details/6595998)

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <errno.h>
5. #include <string.h>
6. #include <sys/types.h>
7. #include <sys/socket.h>
8. #include <netinet/in.h>
9. #include <arpa/inet.h>
11. #define MYPORT 88960    // the port users will be connecting to
13. #define BACKLOG 10     // how many pending connections queue will hold
15. #define BUF\_SIZE 200
17. **int** fd\_A[BACKLOG];    // accepted connection fd
18. **int** conn\_amount;    // current connection amount
20. **void** showclient()
21. {
22. **int** i;
23. printf("client amount: %d\n", conn\_amount);
24. **for** (i = 0; i < BACKLOG; i++) {
25. printf("[%d]:%d  ", i, fd\_A[i]);
26. }
27. printf("\n\n");
28. }
30. **int** main(**void**)
31. {
32. **int** sock\_fd, new\_fd;  // listen on sock\_fd, new connection on new\_fd
33. **struct** sockaddr\_in server\_addr;    // server address information
34. **struct** sockaddr\_in client\_addr; // connector's address information
35. socklen\_t sin\_size;
36. **int** yes = 1;
37. **char** buf[BUF\_SIZE];
38. **int** ret;
39. **int** i;
41. **if** ((sock\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {
42. perror("socket");
43. exit(1);
44. }
46. **if** (setsockopt(sock\_fd, SOL\_SOCKET, SO\_REUSEADDR, &yes, **sizeof**(**int**)) == -1) {
47. perror("setsockopt");
48. exit(1);
49. }
51. server\_addr.sin\_family = AF\_INET;         // host byte order
52. server\_addr.sin\_port = htons(MYPORT);     // short, network byte order
53. server\_addr.sin\_addr.s\_addr = INADDR\_ANY; // automatically fill with my IP
54. memset(server\_addr.sin\_zero, '\0', **sizeof**(server\_addr.sin\_zero));
56. **if** (bind(sock\_fd, (**struct** sockaddr \*)&server\_addr, **sizeof**(server\_addr)) == -1) {
57. perror("bind");
58. exit(1);
59. }
61. **if** (listen(sock\_fd, BACKLOG) == -1) {
62. perror("listen");
63. exit(1);
64. }
66. printf("listen port %d\n", MYPORT);
68. fd\_set fdsr;
69. **int** maxsock;
70. **struct** timeval tv;
72. conn\_amount = 0;
73. sin\_size = **sizeof**(client\_addr);
74. maxsock = sock\_fd;
75. **while** (1) {
76. // initialize file descriptor set
77. FD\_ZERO(&fdsr);
78. FD\_SET(sock\_fd, &fdsr);
80. // timeout setting
81. tv.tv\_sec = 30;
82. tv.tv\_usec = 0;
84. // add active connection to fd set
85. **for** (i = 0; i < BACKLOG; i++) {
86. **if** (fd\_A[i] != 0) {
87. FD\_SET(fd\_A[i], &fdsr);
88. }
89. }
91. ret = select(maxsock + 1, &fdsr, NULL, NULL, &tv);
92. **if** (ret < 0) {
93. perror("select");
94. **break**;
95. } **else** **if** (ret == 0) {
96. printf("timeout\n");
97. **continue**;
98. }
100. // check every fd in the set
101. **for** (i = 0; i < conn\_amount; i++) {
102. **if** (FD\_ISSET(fd\_A[i], &fdsr)) {
103. ret = recv(fd\_A[i], buf, **sizeof**(buf), 0);
105. **char** str[] = "Good,very nice!\n";
107. send(fd\_A[i],str,**sizeof**(str) + 1, 0);

110. **if** (ret <= 0) {        // client close
111. printf("client[%d] close\n", i);
112. close(fd\_A[i]);
113. FD\_CLR(fd\_A[i], &fdsr);
114. fd\_A[i] = 0;
115. } **else** {        // receive data
116. **if** (ret < BUF\_SIZE)
117. memset(&buf[ret], '\0', 1);
118. printf("client[%d] send:%s\n", i, buf);
119. }
120. }
121. }
123. // check whether a new connection comes
124. **if** (FD\_ISSET(sock\_fd, &fdsr)) {
125. new\_fd = accept(sock\_fd, (**struct** sockaddr \*)&client\_addr, &sin\_size);
126. **if** (new\_fd <= 0) {
127. perror("accept");
128. **continue**;
129. }
131. // add to fd queue
132. **if** (conn\_amount < BACKLOG) {
133. fd\_A[conn\_amount++] = new\_fd;
134. printf("new connection client[%d] %s:%d\n", conn\_amount,
135. inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));
136. **if** (new\_fd > maxsock)
137. maxsock = new\_fd;
138. }
139. **else** {
140. printf("max connections arrive, exit\n");
141. send(new\_fd, "bye", 4, 0);
142. close(new\_fd);
143. **break**;
144. }
145. }
146. showclient();
147. }
149. // close other connections
150. **for** (i = 0; i < BACKLOG; i++) {
151. **if** (fd\_A[i] != 0) {
152. close(fd\_A[i]);
153. }
154. }
156. exit(0);
157. }

补充部分：

## 1 基本原理



注：select 原理图，摘自 [*IBM iSeries 信息中心*](http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=%2Frzab6%2Frzab6xnonblock.htm)。

## 1 数据结构与函数原型

### 1.1 select

* 函数原型
* int select(
* int nfds,
* fd\_set \*readset,
* fd\_set \*writeset,
* fd\_set\* exceptset,
* struct timeval \*timeout
* );
* 头文件
  + select位于：
  + #include <sys/select.h>
  + struct timeval位于：
  + #include <sys/time.h>
* 返回值

返回对应位仍然为1的fd的总数。

* 参数
  + nfds：第一个参数是：最大的文件描述符值+1；
  + readset：可读描述符集合；
  + writeset：可写描述符集合；
  + exceptset：异常描述符；
  + timeout：select 的监听时长，如果这短时间内所监听的 socket 没有事件发生。

### 1.2 fd\_set

#### 1.2.1 清空描述符集合

FD\_ZERO(fd\_set \*)

#### 1.2.2 向描述符集合添加指定描述符

FD\_SET(int, fd\_set \*)

#### 1.2.3 从描述符集合删除指定描述符

FD\_CLR(int, fd\_set \*)

#### 1.2.4 检测指定描述符是否在描述符集合中

FD\_ISSET(int, fd\_set \*)

#### 1.2.5 描述符最大数量

#define FD\_SETSIZE 1024

### 1.3 描述符集合

可读描述符集合中可读的描述符，为1，其他为0；可写也类似。异常描述符集合中有异常等待处理的描述符的值为1，其他为0。

### 1.4 ioctl

* 函数原型：
* int ioctl(int handle, int cmd,[int \*argdx, int argcx]);
* 头文件：
* #include <sys/ioctl.h>
* 返回值：
  + 0 - 成功
  + 1 - 失败

## 2 示例

程序各部分的解释在注释中。

#include <sys/socket.h>

#include <string.h>

#include <sys/time.h>

#include <netinet/in.h>

#include <sys/ioctl.h>

#include <stdlib.h>

#include <errno.h>

#include <stdio.h>

#include <unistd.h>

#define TRUE 1

#define FALSE 0

int main(int argc, char \*argv[])

{

int i, len, rc, on = TRUE;

int listen\_sd, new\_sd = 0, max\_sd;

int desc\_ready;

char buffer[80];

int close\_conn, end\_server = FALSE;

struct sockaddr\_in server\_addr;

struct timeval timeout;

struct fd\_set master\_set, working\_set;

// Listen

listen\_sd = socket(AF\_INET, SOCK\_STREAM, 0);

if (listen\_sd < 0)

{

perror("socket() failed");

exit(-1);

}

// Set socket options

rc = setsockopt(listen\_sd, SOL\_SOCKET, SO\_REUSEADDR, (char \*) &on, sizeof(on));

if (rc < 0)

{

perror("setsockopt() failed");

close(listen\_sd);

exit(-1);

}

// Set IO control

rc = ioctl(listen\_sd, FIONBIO, (char \*) &on);

if (rc < 0)

{

perror("ioctl() failed");

close(listen\_sd);

exit(-1);

}

// Bind

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

server\_addr.sin\_port = htons(atoi(argv[1]));

rc = bind(listen\_sd, (struct sockaddr \*) &server\_addr, sizeof(server\_addr));

if (rc < 0)

{

perror("bind() failed\n");

close(listen\_sd);

exit(-1);

}

// Listen

rc = listen(listen\_sd, 32);

if (rc < 0)

{

perror("listen() failed\n");

close(listen\_sd);

exit(-1);

}

// Intialize sd set

FD\_ZERO(&master\_set);

max\_sd = listen\_sd;

FD\_SET(listen\_sd, &master\_set);

timeout.tv\_sec = 3 \* 60;

timeout.tv\_usec = 0;

// Start

do

{

// Copy master\_set into working\_set

memcpy(&working\_set, &master\_set, sizeof(master\_set));

printf("Waiting on select()...\n");

rc = select(max\_sd + 1, &working\_set, NULL, NULL, &timeout);

if (rc < 0)

{

perror(" select() failed\n");

break;

}

if (rc == 0)

{

printf(" select() timed out. End program.\n");

break;

}

desc\_ready = rc; // number of sds ready in working\_set

// Check each sd in working\_set

for (i = 0; i <= max\_sd && desc\_ready > 0; ++i)

{

// Check to see if this sd is ready

if (FD\_ISSET(i, &working\_set))

{

--desc\_ready;

// Check to see if this is the listening sd

if (i == listen\_sd)

{

printf(" Listeing socket is readable\n");

do

{

// Accept

new\_sd = accept(listen\_sd, NULL, NULL);

// Nothing to be accepted

if (new\_sd < 0)

{

// All have been accepted

if (errno != EWOULDBLOCK)

{

perror(" accept() failed\n");

end\_server = TRUE;

}

break;

}

// Insert new\_sd into master\_set

printf(" New incoming connection - %d\n", new\_sd);

FD\_SET(new\_sd, &master\_set);

if (new\_sd > max\_sd)

{

max\_sd = new\_sd;

}

}

while (new\_sd != -1);

}

// This is not the listening sd

else

{

close\_conn = FALSE;

printf(" Descriptor %d is avaliable\n", i);

do

{

rc = recv(i, buffer, sizeof(buffer), 0);

// Receive data on sd "i", until failure occurs

if (rc < 0)

{

// Normal failure

if (errno != EWOULDBLOCK)

{

perror(" recv() failed\n");

close\_conn = TRUE;

}

break;

}

// The connection has been closed by the client

if (rc == 0)

{

printf(" Connection closed\n");

close\_conn = TRUE;

break;

}

/\* Receiving data succeeded and echo it back

the to client \*/

len = rc;

printf(" %d bytes received\n", len);

rc = send(i, buffer, len, 0);

if (rc < 0)

{

perror(" send() failed");

close\_conn = TRUE;

break;

}

}

while (TRUE);

// If unknown failure occured

if (close\_conn)

{

// Close the sd and remove it from master\_set

close(i);

FD\_CLR(i, &master\_set);

// If this is the max sd

if (i == max\_sd)

{

// Find the max sd in master\_set now

while (FD\_ISSET(max\_sd, &master\_set) == FALSE)

{

--max\_sd;

}

} // End of if (i == max\_sd)

} // End of if (close\_conn)

}

}

}

}

while (end\_server == FALSE);

/\* Close each sd in master\_set \*/

for (i = 0; i < max\_sd; ++i)

{

if (FD\_ISSET(i, &master\_set))

{

close(i);

}

}

return 0;

}