<https://blog.csdn.net/coder_yi_liu/article/details/8249421>

1.select 模型是一个比较传统的异步IO模型，我们知道的著名的apache就是基于select模型，而我之前工作过的搜狐畅游的天龙八部，还有几款游戏都是基于select模型。

对于select模型，大多都是说他的缺点，实际上我的观点有点不一样，select模型的跨平台性是比较好的，开发也比较简单，只有当个进程连接数的限制，以及其性能随着连接数增长下降的问题，实际上都得根据项目的实际情况而定的。在内部分布式通讯中，几乎所有的连接都是活跃的情况下，select模型并不比epoll的性能差，只是在一些应用中大部分连接都不活跃的情况，epoll的使用效果要高，可以使单台服务器的承载量大大增加，实际上游戏服务器中玩家绝大部发是活跃的，因此实际性能也还不错。

下面是我自己写的一个简单的select模型的demo，其中要注意两个地方，估计写过select的同学很多都遇到类似的问题，一个是超时的变量实际上每次select后需要重新赋值，否则可能出现一只超时的情况， 二就是fd\_set 每次select后，不活跃的会清零，需要一个备份的。再个就是保存客户端socket需要一个数组，队列，或者其他数据结构都可以，要注意增加，清除的问题。

**[cpp]** [view plain](https://blog.csdn.net/coder_yi_liu/article/details/8249421) [copy](https://blog.csdn.net/coder_yi_liu/article/details/8249421)

1. #include<iostream>
2. #include<stdio.h>
3. #include<stdlib.h>
4. #include<unistd.h>
5. #include<errno.h>
6. #include<netdb.h>
7. #include<sys/types.h>
8. #include<sys/socket.h>
9. #include<netinet/in.h>
10. #include<arpa/inet.h>
11. #include<netdb.h>
12. #include<sys/time.h>
13. #include<string.h>
14. #include<sys/select.h>
15. #include<pthread.h>
16. **using** **namespace** std;
18. **int** max\_fd(**int** a[], **int** n)
19. {
20. **int** max = 0;
21. **for**(**int** i = 0; i < n; i++)
22. {
23. **if**(max < a[i])
24. {
25. max = a[i];
26. }
27. }
29. **return**  max;
30. }
32. **int** main(**int** argc, **char**\*argv[])
33. {
34. **int** port = 0;
35. **int** N = 0;
36. **if** (argc != 3)
37. {
38. cout<<"command error"<<endl;
39. exit(-1);
40. }
42. port = atoi(argv[1]);
43. N = atoi(argv[2]);
44. **if**(N > FD\_SETSIZE)
45. {
46. N =   FD\_SETSIZE;
47. }
48. **int** server\_sock = 0;
49. **struct** sockaddr\_in server\_addr;
50. memset(&server\_addr, 0, **sizeof**(server\_addr));
52. **if**((server\_sock = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)
53. {
54. cout<<"create socket error"<<endl;
55. exit(-1);
56. }
58. server\_addr.sin\_family = AF\_INET;
59. server\_addr.sin\_addr.s\_addr =htonl(INADDR\_ANY);
60. server\_addr.sin\_port = htons(port);
62. **if**(bind(server\_sock, (**struct** sockaddr\*)&server\_addr,**sizeof**(sockaddr))
63. == -1)
64. {
65. cout<<"bind error"<<endl;
66. exit(-1);
67. }
69. **if**(listen(server\_sock, 5) == -1)
70. {
71. cout<<"listent error"<<endl;
72. exit(-1);
73. }
75. fd\_set fd[2];
76. FD\_ZERO(&fd[0]);
77. FD\_SET(server\_sock, &fd[0]);
78. **int** \*sock = **new** **int**[N];
79. memset(sock, 0, **sizeof**(**int**)\*N);
80. sock[0] = server\_sock;
81. **int** count = 0;
83. **while**(1)
84. {
85. **struct** timeval tv = {5, 0};
86. FD\_ZERO(&fd[1]);
87. fd[1] = fd[0];
88. **int** ret = select(max\_fd(sock, N)+1, &fd[1], NULL, NULL, &tv);
89. **if**(ret < 0)
90. {
91. cout<<"select error"<<endl;
92. }
93. **else** **if**(ret == 0)
94. {
95. cout<<"time out"<<endl;
96. }
97. **else**
98. {
99. **if**(FD\_ISSET(sock[0], &fd[1]) && count < N-1)
100. {
101. **struct** sockaddr\_in client\_addr;
102. memset(&client\_addr, 0, **sizeof**(client\_addr));
103. unsigned **int** len = **sizeof**(client\_addr);
104. **int** new\_sock=accept(sock[0],(**struct** sockaddr\*)&client\_addr, &len);
105. **if**(new\_sock == -1)
106. {
107. cout<<"accept error"<<endl;
108. }
109. **else**
110. {
111. **for**(**int** i = 1; i < N; i++)
112. {
113. **if**(sock[i] == 0)
114. {
115. sock[i] = new\_sock;
116. FD\_SET(new\_sock, &fd[0]);
117. count++;
118. **break**;
119. }
120. }
121. }
123. }
125. **char** recvbuf[1024] = {0};
126. **char** sendbuf[1024] = {0};
127. **for**(**int** i = 1; i < N; i++)
128. {
129. **if**(FD\_ISSET(sock[i], &fd[1]))
130. {
131. **if**(recv(sock[i], recvbuf, **sizeof**(recvbuf), 0) <= 0)
132. {
133. cout<<"recv error"<<endl;
134. FD\_CLR(sock[i], &fd[0]);
135. close(sock[i]);
136. sock[i] = 0;
137. count--;
138. **continue**;
139. }
141. strcpy(sendbuf, recvbuf);
143. **if**(send(sock[i], sendbuf, **sizeof**(sendbuf), 0) <= 0)
144. {
145. cout<<"send error"<<endl;
146. FD\_CLR(sock[i], &fd[0]);
147. close(sock[i]);
148. sock[i] = 0;
149. count--;
150. **continue**;
151. }
152. }
154. }//end for
156. }
158. }//end while
159. **return** 0;
161. }