<https://blog.csdn.net/weiyuefei/article/details/52242890>

**6.1 ET模式为什么要设置在非阻塞模式下工作**

    因为ET模式下的读写需要一直读或写直到出错（对于读，当读到的实际字节数小于请求字节数时就可以停止），而如果你的文件描述符如果不是非阻塞的，那这个一直读或一直写势必会在最后一次阻塞。这样就不能在阻塞在epoll\_wait上了，造成其他文件描述符的任务饿死。

**6.2 使用ET和LT的区别**

LT：水平触发，效率会低于ET触发，尤其在大并发，大流量的情况下。但是LT对代码编写要求比较低，不容易出现问题。LT模式服务编写上的表现是：只要有数据没有被获取，内核就不断通知你，因此不用担心事件丢失的情况。

ET：边缘触发，效率非常高，在并发，大流量的情况下，**会比LT少很多epoll的系统调用**，因此效率高。但是对编程要求高，需要细致的处理每个请求，否则容易发生丢失事件的情况。

下面举一个列子来说明LT和ET的区别（都是非阻塞模式，阻塞就不说了，效率太低）：

采用LT模式下，如果accept调用有返回就可以马上建立当前这个连接了，再epoll\_wait等待下次通知，和select一样。

但是对于ET而言，如果accpet调用有返回，除了建立当前这个连接外，不能马上就epoll\_wait还需要继续循环accpet，直到返回-1，且errno==EAGAIN，

**从本质上讲：与LT相比，ET模型是通过减少系统调用来达到提高并行效率的。**

**6.3 一道腾讯后台开发的面试题**

    使用Linux epoll模型，**水平（LT）触发模式**，当socket可写时，会不停的触发socket可写的事件，如何处理？

第一种最普遍的方式：  
需要向socket写数据的时候才把socket加入epoll，等待可写事件。接受到可写事件后，调用write或者send发送数据。当所有数据都写完后，把socket移出epoll。

这种方式的缺点是，即使发送很少的数据，也要把socket加入epoll，写完后在移出epoll，有一定操作代价。

一种改进的方式：  
开始不把socket加入epoll，需要向socket写数据的时候，直接调用write或者send发送数据。如果返回EAGAIN，把socket加入epoll，在epoll的驱动下写数据，全部数据发送完毕后，再移出epoll。

这种方式的优点是：数据不多的时候可以避免epoll的事件处理，提高效率。

**6.4什么情况下用ET**

很简单，当你想提高程序效率的时候。  
  
最后附一个epoll实例：

点击(此处)折叠或打开

1. #include <sys/socket.h>
2. #include <sys/wait.h>
3. #include <netinet/in.h>
4. #include <netinet/tcp.h>
5. #include <sys/epoll.h>
6. #include <sys/sendfile.h>
7. #include <sys/stat.h>
8. #include <unistd.h>
9. #include <stdio.h>
10. #include <stdlib.h>
11. #include <string.h>
12. #include <strings.h>
13. #include <fcntl.h>
14. #include <errno.h>
15. #define MAX\_EVENTS 10
16. #define PORT 8080
17. //设置socket连接为非阻塞模式
18. void setnonblocking(int sockfd) {
19. int opts;
20. opts = fcntl(sockfd, F\_GETFL);
21. if(opts < 0) {
22. perror("fcntl(F\_GETFL)\n");
23. exit(1);
24. }
25. opts = (opts | O\_NONBLOCK);
26. if(fcntl(sockfd, F\_SETFL, opts) < 0) {
27. perror("fcntl(F\_SETFL)\n");
28. exit(1);
29. }
30. }
31. int main(){
32. struct epoll\_event ev, events[MAX\_EVENTS]; //ev负责添加事件，events接收返回事件
33. int addrlen, listenfd, conn\_sock, nfds, epfd, fd, i, nread, n;
34. struct sockaddr\_in local, remote;
35. char buf[BUFSIZ];
36. //创建listen socket
37. if( (listenfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {
38. perror("sockfd\n");
39. exit(1);
40. }
41. setnonblocking(listenfd);//listenfd设置为非阻塞[1]
42. bzero(&local, sizeof(local));
43. local.sin\_family = AF\_INET;
44. local.sin\_addr.s\_addr = htonl(INADDR\_ANY);;
45. local.sin\_port = htons(PORT);
46. if( bind(listenfd, (struct sockaddr \*) &local, sizeof(local)) < 0) {
47. perror("bind\n");
48. exit(1);
49. }
50. listen(listenfd, 20);
51. epfd = epoll\_create(MAX\_EVENTS);
52. if (epfd == -1) {
53. perror("epoll\_create");
54. exit(EXIT\_FAILURE);
55. }
56. ev.events = EPOLLIN;
57. ev.data.fd = listenfd;
58. if (epoll\_ctl(epfd, EPOLL\_CTL\_ADD, listenfd, &ev) == -1) {//监听listenfd
59. perror("epoll\_ctl: listen\_sock");
60. exit(EXIT\_FAILURE);
61. }
62. for (;;) {
63. nfds = epoll\_wait(epfd, events, MAX\_EVENTS, -1);
64. if (nfds == -1) {
65. perror("epoll\_pwait");
66. exit(EXIT\_FAILURE);
67. }
68. for (i = 0; i < nfds; ++i) {
69. fd = events[i].data.fd;
70. if (fd == listenfd) {
71. while ((conn\_sock = accept(listenfd,(struct sockaddr \*) &remote,
72. (size\_t \*)&addrlen)) > 0) {
73. setnonblocking(conn\_sock);//下面设置ET模式，所以要设置非阻塞
74. ev.events = EPOLLIN | EPOLLET;
75. ev.data.fd = conn\_sock;
76. if (epoll\_ctl(epfd, EPOLL\_CTL\_ADD, conn\_sock, &ev) == -1) {//读监听
77. perror("epoll\_ctl: add"); //连接套接字
78. exit(EXIT\_FAILURE);
79. }
80. }
81. if (conn\_sock == -1) {
82. if (errno != EAGAIN && errno != ECONNABORTED
83. && errno != EPROTO && errno != EINTR)
84. perror("accept");
85. }
86. continue;
87. }
88. if (events[i].events & EPOLLIN) {
89. n = 0;
90. while ((nread = read(fd, buf + n, BUFSIZ-1)) > 0) {//ET下可以读就一直读
91. n += nread;
92. }
93. if (nread == -1 && errno != EAGAIN) {
94. perror("read error");
95. }
96. ev.data.fd = fd;
97. ev.events = events[i].events | EPOLLOUT; //MOD OUT
98. if (epoll\_ctl(epfd, EPOLL\_CTL\_MOD, fd, &ev) == -1) {
99. perror("epoll\_ctl: mod");
100. }
101. }
102. if (events[i].events & EPOLLOUT) {
103. sprintf(buf, "HTTP/1.1 200 OK\r\nContent-Length: %d\r\n\r\nHello World", 11);
104. int nwrite, data\_size = strlen(buf);
105. n = data\_size;
106. while (n > 0) {
107. nwrite = write(fd, buf + data\_size - n, n);//ET下一直将要写数据写完
108. if (nwrite < n) {
109. if (nwrite == -1 && errno != EAGAIN) {
110. perror("write error");
111. }
112. break;
113. }
114. n -= nwrite;
115. }
116. close(fd);
117. }
118. }
119. }
120. return 0;
121. }