code/conc/echoservers.c

```
1  void check_clients(pool *p)
2  {
3      int i, connfd, n;
4      char buf[MAXLINE];
5      rio_t rio;
6
7      for (i = 0; (i <= p->maxi) && (p->nready > 0); i++) {
8          connfd = p->clientfd[i];
9          rio = p->clientrio[i];
10
11         /* If the descriptor is ready, echo a text line from it */
12         if ((connfd > 0) && (FD_ISSET(connfd, &p->ready_set))) {
13             p->nready--;
14             if ((n = Rio_readlineb(&rio, buf, MAXLINE)) != 0) {
15                 byte_cnt += n;
16                 printf("Server received %d (%d total) bytes on fd %d\n",
17                        n, byte_cnt, connfd);
18                 Rio_writen(connfd, buf, n);
19             }
20
21             /* EOF detected, remove descriptor from pool */
22             else {
23                 Close(connfd);
24                 FD_CLR(connfd, &p->read_set);
25                 p->clientfd[i] = -1;
26             }
27         }
28     }
29 }
```

code/conc/echoservers.c

图 12-11　check_clients 服务准备好的客户端连接

12.2.2　I/O 多路复用技术的优劣

图 12-8 中的服务器提供了一个很好的基于 I/O 多路复用的事件驱动编程的优缺点示例。事件驱动设计的一个优点是，它比基于进程的设计给了程序员更多的对程序行为的控制。例如，我们可以设想编写一个事件驱动的并发服务器，为某些客户端提供它们需要的服务，而这对于基于进程的并发服务器来说，是很困难的。

另一个优点是，一个基于 I/O 多路复用的事件驱动服务器是运行在单一进程上下文中的，因此每个逻辑流都能访问该进程的全部地址空间。这使得在流之间共享数据变得很容易。一个与作为单个进程运行相关的优点是，你可以利用熟悉的调试工具，例如 GDB，来调试你的并发服务器，就像对顺序程序那样。最后，事件驱动设计常常比基于进程的设计要高效得多，因为它们不需要进程上下文切换来调度新的流。

事件驱动设计一个明显的缺点就是编码复杂。我们的事件驱动的并发 echo 服务器需要的代码比基于进程的服务器多三倍，并且很不幸，随着并发粒度的减小，复杂性还会上升。这里的*粒度*是指每个逻辑流每个时间片执行的指令数量。例如，在示例并发服务器中，并发粒度就是读一个完整的文本行所需要的指令数量。只要某个逻辑流正忙于读一个文本行，其他逻辑流就不可能有进展。对我们的例子来说这没有问题，但是它使得在“故意只发送部分文