

后，我们将这个已连接描述符添加到 select 读集合(第 12 行)，并更新该池的一些全局属性。maxfd 变量(第 15~16 行)记录了 select 的最大文件描述符。maxi 变量(第 17~18 行)记录的是到 clientfd 数组的最大索引，这样 check\_clients 函数就无需搜索整个数组了。

```


code/conc/echoservers.c
1 void add_client(int connfd, pool *p)
2 {
3     int i;
4     p->nready--;
5     for (i = 0; i < FD_SETSIZE; i++) /* Find an available slot */
6         if (p->clientfd[i] < 0) {
7             /* Add connected descriptor to the pool */
8             p->clientfd[i] = connfd;
9             Rio_readinitb(&p->clientrio[i], connfd);
10
11             /* Add the descriptor to descriptor set */
12             FD_SET(connfd, &p->read_set);
13
14             /* Update max descriptor and pool high water mark */
15             if (connfd > p->maxfd)
16                 p->maxfd = connfd;
17             if (i > p->maxi)
18                 p->maxi = i;
19             break;
20         }
21     if (i == FD_SETSIZE) /* Couldn't find an empty slot */
22         app_error("add_client error: Too many clients");
23 }
code/conc/echoservers.c

```

图 12-10 add\_client 向池中添加一个新的客户端连接

图 12-11 中的 check\_clients 函数回送来自每个准备好的已连接描述符的一个文本行。如果成功地从描述符读取了一个文本行，那么就该文本行回送到客户端(第 15~18 行)。注意，在第 15 行我们维护着一个从所有客户端接收到的全部字节的累计值。如果因为客户端关闭这个连接中它的那一端，检测到 EOF，那么将关闭这边的连接端(第 23 行)，并从池中清除掉这个描述符(第 24~25 行)。

根据图 12-7 中的有限状态模型，select 函数检测到输入事件，而 add\_client 函数创建一个新的逻辑流(状态机)。check\_clients 函数回送输入行，从而执行状态转移，而且当客户端完成文本行发送时，它还要删除这个状态机。

 **练习题 12.4** 图 12-8 所示的服务器中，我们在每次调用 select 之前都立即小心地重新初始化 pool.ready\_set 变量。为什么？

### 旁注 事件驱动的 Web 服务器

尽管有 12.2.2 节中说明的缺点，现代高性能服务器(例如 Node.js、nginx 和 Tor-nado)使用的都是基于 I/O 多路复用的事件驱动的编程方式，主要是因为相比于进程和线程的方式，它有明显的性能优势。