code/conc/echo-cnt.c

```
1   #include "csapp.h"
2
3   static int byte_cnt;  /* Byte counter */
4   static sem_t mutex;   /* and the mutex that protects it */
5
6   static void init_echo_cnt(void)
7   {
8       Sem_init(&mutex, 0, 1);
9       byte_cnt = 0;
10  }
11
12  void echo_cnt(int connfd)
13  {
14      int n;
15      char buf[MAXLINE];
16      rio_t rio;
17      static pthread_once_t once = PTHREAD_ONCE_INIT;
18
19      Pthread_once(&once, init_echo_cnt);
20      Rio_readinitb(&rio, connfd);
21      while((n = Rio_readlineb(&rio, buf, MAXLINE)) != 0) {
22          P(&mutex);
23          byte_cnt += n;
24          printf("server received %d (%d total) bytes on fd %d\n",
25                 n, byte_cnt, connfd);
26          V(&mutex);
27          Rio_writen(connfd, buf, n);
28      }
29  }
```

code/conc/echo-cnt.c

图 12-29　echo_cnt：echo 的一个版本，它对从客户端接收的所有字节计数

一旦程序包被初始化，echo_cnt 函数会初始化 RIO 带缓冲区的 I/O 包(第 20 行)，然后回送从客户端接收到的每一个文本行。注意，在第 23～25 行中对共享变量 byte_cnt 的访问是被 P 和 V 操作保护的。

旁注　基于线程的事件驱动程序

I/O 多路复用不是编写事件驱动程序的唯一方法。例如，你可能已经注意到我们刚才开发的并发的预线程化的服务器实际上是一个事件驱动服务器，带有主线程和工作者线程的简单状态机。主线程有两种状态(“等待连接请求”和“等待可用的缓冲区槽位”)、两个 I/O 事件(“连接请求到达”和“缓冲区槽位变为可用”)和两个转换(“接受连接请求”和“插入缓冲区项目”)。类似地，每个工作者线程有一个状态(“等待可用的缓冲项目”)、一个 I/O 事件(“缓冲区项目变为可用”)和一个转换(“取出缓冲区项目”)。

12.6　使用线程提高并行性

到目前为止，在对并发的研究中，我们都假设并发线程是在单处理器系统上执行的。