12.1.1 基于进程的并发服务器

图 12-5 展示了一个基于进程的并发 echo 服务器的代码。第 29 行调用的 echo 函数来自于图 11-21。关于这个服务器，有几点重要内容需要说明：

- 首先，通常服务器会运行很长的时间，所以我们必须要包括一个 SIGCHLD 处理程序，来回收僵死(zombie)子进程的资源(第 4～9 行)。因为当 SIGCHLD 处理程序执行时，SIGCHLD 信号是阻塞的，而 Linux 信号是不排队的，所以 SIGCHLD 处理程序必须准备好回收多个僵死子进程的资源。
- 其次，父子进程必须关闭它们各自的 connfd(分别为第 33 行和第 30 行)副本。就像我们已经提到过的，这对父进程而言尤为重要，它必须关闭它的已连接描述符，以避免内存泄漏。
- 最后，因为套接字的文件表表项中的引用计数，直到父子进程的 connfd 都关闭了，到客户端的连接才会终止。

code/conc/echoserverp.c

```
1  #include "csapp.h"
2  void echo(int connfd);
3
4  void sigchld_handler(int sig)
5  {
6      while (waitpid(-1, 0, WNOHANG) > 0)
7          ;
8      return;
9  }
10
11 int main(int argc, char **argv)
12 {
13     int listenfd, connfd;
14     socklen_t clientlen;
15     struct sockaddr_storage clientaddr;
16
17     if (argc != 2) {
18         fprintf(stderr, "usage: %s <port>\n", argv[0]);
19         exit(0);
20     }
21
22     Signal(SIGCHLD, sigchld_handler);
23     listenfd = Open_listenfd(argv[1]);
24     while (1) {
25         clientlen = sizeof(struct sockaddr_storage);
26         connfd = Accept(listenfd, (SA *) &clientaddr, &clientlen);
27         if (Fork() == 0) {
28             Close(listenfd); /* Child closes its listening socket */
29             echo(connfd);    /* Child services client */
30             Close(connfd);   /* Child closes connection with client */
31             exit(0);         /* Child exits */
32         }
33         Close(connfd); /* Parent closes connected socket (important!) */
34     }
35 }
```

code/conc/echoserverp.c

图 12-5 基于进程的并发 echo 服务器。父进程派生一个子进程来处理每个新的连接请求