要了解这样会如何影响正确性，来看一个简单的应用，它本质上类似于像 shell 和 Web 服务器这样的真实程序。基本的结构是父进程创建一些子进程，这些子进程各自独立运行一段时间，然后终止。父进程必须回收子进程以避免在系统中留下僵死进程。但是我们还希望父进程能够在子进程运行时自由地去做其他的工作。所以，我们决定用 SIGCHLD 处理程序来回收子进程，而不是显式地等待子进程终止。（回想一下，只要有一个子进程终止或者停止，内核就会发送一个 SIGCHLD 信号给父进程。）

图 8-36 展示了我们的初次尝试。父进程设置了一个 SIGCHLD 处理程序，然后创建

code/ecf/signal1.c

```
1  /* WARNING: This code is buggy! */
2
3  void handler1(int sig)
4  {
5      int olderrno = errno;
6
7      if ((waitpid(-1, NULL, 0)) < 0)
8          sio_error("waitpid error");
9      Sio_puts("Handler reaped child\n");
10     Sleep(1);
11     errno = olderrno;
12 }
13
14 int main()
15 {
16     int i, n;
17     char buf[MAXBUF];
18
19     if (signal(SIGCHLD, handler1) == SIG_ERR)
20         unix_error("signal error");
21
22     /* Parent creates children */
23     for (i = 0; i < 3; i++) {
24         if (Fork() == 0) {
25             printf("Hello from child %d\n", (int)getpid());
26             exit(0);
27         }
28     }
29
30     /* Parent waits for terminal input and then processes it */
31     if ((n = read(STDIN_FILENO, buf, sizeof(buf))) < 0)
32         unix_error("read");
33
34     printf("Parent processing input\n");
35     while (1)
36         ;
37
38     exit(0);
39 }
```

code/ecf/signal1.c

图 8-36　signal1:这个程序是有缺陷的，因为它假设信号是排队的