就是正确的。如果它没有，那么结果就是错误的。这样的错误非常难以调试，因为几乎不可能测试所有的交错。你可能运行这段代码十亿次，也没有一次错误，但是下一次测试却导致引发竞争的交错。

code/ecf/procmask1.c

```
1  /* WARNING: This code is buggy! */
2  void handler(int sig)
3  {
4      int olderrno = errno;
5      sigset_t mask_all, prev_all;
6      pid_t pid;
7
8      Sigfillset(&mask_all);
9      while ((pid = waitpid(-1, NULL, 0)) > 0) { /* Reap a zombie child */
10         Sigprocmask(SIG_BLOCK, &mask_all, &prev_all);
11         deletejob(pid); /* Delete the child from the job list */
12         Sigprocmask(SIG_SETMASK, &prev_all, NULL);
13     }
14     if (errno != ECHILD)
15         Sio_error("waitpid error");
16     errno = olderrno;
17 }
18
19 int main(int argc, char **argv)
20 {
21     int pid;
22     sigset_t mask_all, prev_all;
23
24     Sigfillset(&mask_all);
25     Signal(SIGCHLD, handler);
26     initjobs(); /* Initialize the job list */
27
28     while (1) {
29         if ((pid = Fork()) == 0) { /* Child process */
30             Execve("/bin/date", argv, NULL);
31         }
32         Sigprocmask(SIG_BLOCK, &mask_all, &prev_all); /* Parent process */
33         addjob(pid);  /* Add the child to the job list */
34         Sigprocmask(SIG_SETMASK, &prev_all, NULL);
35     }
36     exit(0);
37 }
```

code/ecf/procmask1.c

图 8-39　一个具有细微同步错误的 shell 程序。如果子进程在父进程能够开始运行前就结束了，那么 addjob 和 deletejob 会以错误的方式被调用

图 8-40 展示了消除图 8-39 中竞争的一种方法。通过在调用 fork 之前，阻塞 SIGCHLD 信号，然后在调用 addjob 之后取消阻塞这些信号，我们保证了在子进程被添加到作业列表中之后回收该子进程。注意，子进程继承了它们父进程的被阻塞集合，所以我们必须在调用 execve 之前，小心地解除子进程中阻塞的 SIGCHLD 信号。