进入一个无限循环。它阻塞 SIGCHLD 信号，避免 8.5.6 节中讨论过的父进程和子进程之间的竞争。创建了子进程之后，把 pid 重置为 0，取消阻塞 SIGCHLD，然后以循环的方式等待 pid 变为非零。子进程终止后，处理程序回收它，把它非零的 PID 赋值给全局 pid 变量。这会终止循环，父进程继续其他的工作，然后开始下一次迭代。

code/ecf/waitforsignal.c

```
1  #include "csapp.h"
2
3  volatile sig_atomic_t pid;
4
5  void sigchld_handler(int s)
6  {
7      int olderrno = errno;
8      pid = waitpid(-1, NULL, 0);
9      errno = olderrno;
10 }
11
12 void sigint_handler(int s)
13 {
14 }
15
16 int main(int argc, char **argv)
17 {
18     sigset_t mask, prev;
19
20     Signal(SIGCHLD, sigchld_handler);
21     Signal(SIGINT, sigint_handler);
22     Sigemptyset(&mask);
23     Sigaddset(&mask, SIGCHLD);
24
25     while (1) {
26         Sigprocmask(SIG_BLOCK, &mask, &prev); /* Block SIGCHLD */
27         if (Fork() == 0) /* Child */
28             exit(0);
29
30         /* Parent */
31         pid = 0;
32         Sigprocmask(SIG_SETMASK, &prev, NULL); /* Unblock SIGCHLD */
33
34         /* Wait for SIGCHLD to be received (wasteful) */
35         while (!pid)
36             ;
37
38         /* Do some work after receiving SIGCHLD */
39         printf(".");
40     }
41     exit(0);
42 }
```

code/ecf/waitforsignal.c

图 8-41　用循环来等待信号。这段代码正确，但循环是一种浪费