```
10  }
11
12  int main()
13  {
14      int i;
15
16      Signal(SIGUSR2, handler);
17
18      if (Fork() == 0) {  /* Child */
19          for (i = 0; i < 5; i++) {
20              Kill(getppid(), SIGUSR2);
21              printf("sent SIGUSR2 to parent\n");
22          }
23          exit(0);
24      }
25
26      Wait(NULL);
27      printf("counter=%d\n", counter);
28      exit(0);
29  }
```

code/ecf/counterprob.c

图 8-45　（续）

**8.24　修改图 8-18 中的程序，以满足下面两个条件：

1）每个子进程在试图写一个只读文本段中的位置时会异常终止。

2）父进程打印和下面所示相同（除了 PID）的输出：

```
child 12255 terminated by signal 11: Segmentation fault
child 12254 terminated by signal 11: Segmentation fault
```

提示：请参考 psignal(3)的 man 页。

**8.25　编写 fgets 函数的一个版本，叫做 tfgets，它 5 秒钟后会超时。tfgets 函数接收和 fgets 相同的输入。如果用户在 5 秒内不键入一个输入行，tfgets 返回 NULL。否则，它返回一个指向输入行的指针。

**8.26　以图 8-23 中的示例作为开始点，编写一个支持作业控制的 shell 程序。shell 必须具有以下特性：

- 用户输入的命令行由一个 name、零个或者多个参数组成，它们都由一个或者多个空格分隔开。如果 name 是一个内置命令，那么 shell 就立即处理它，并等待下一个命令行。否则，shell 就假设 name 是一个可执行文件，在一个初始的子进程（作业）的上下文中加载并运行它。作业的进程组 ID 与子进程的 PID 相同。
- 每个作业是由一个进程 ID（PID）或者一个作业 ID（JID）来标识的，它是由一个 shell 分配的任意的小正整数。JID 在命令行上用前缀“%”来表示。比如，“%5”表示 JID 5，而“5”表示 PID 5。
- 如果命令行以 & 来结束，那么 shell 就在后台运行这个作业。否则，shell 就在前台运行这个作业。
- 输入 Ctrl+C（Ctrl+Z），使得内核发送一个 SIGINT（SIGTSTP）信号给 shell，shell 再转发给前台进程组中的每个进程 ⊖
- 内置命令 jobs 列出所有的后台作业。
- 内置命令 bg *job* 通过发送一个 SIGCONT 信号重启 *job*，然后在后台运行它。*job* 参数可以是一个 PID，也可以是一个 JID。
- 内置命令 fg *job* 通过发送一个 SIGCONT 信号重启 *job*，然后在前台运行它。

⊖ 注意这是对真实的 shell 工作方式的简化。真实的 shell 里，内核响应 Ctrl+C（Ctrl+Z），把 SIGINT（SIGTSTP）直接发送给终端前台进程组中的每个进程。shell 用 tcsetpgrp 函数管理这个进程组的成员，用 tcsetattr 函数管理终端的属性，这两个函数都超出了本书讲述的范围。可以参考[62]获得详细信息。