```
linux> ./restart
starting
processing...
processing...
Ctrl+C
restarting
processing...
Ctrl+C
restarting
processing...
```

关于这个程序有两件很有趣的事情。首先，为了避免竞争，必须在调用了 sigsetjmp 之后再设置处理程序。否则，就会冒在初始调用 sigsetjmp 为 siglongjmp 设置调用环境之前运行处理程序的风险。其次，你可能已经注意到了，sigsetjmp 和 siglongjmp 函数不在图 8-33 中异步信号安全的函数之列。原因是一般来说 siglongjmp 可以跳到任意代码，所以我们必须小心，只在 siglongjmp 可达的代码中调用安全的函数。在本例中，我们调用安全的 sio_puts 和 sleep 函数。不安全的 exit 函数是不可达的。

code/ecf/restart.c

```
1  #include "csapp.h"
2
3  sigjmp_buf buf;
4
5  void handler(int sig)
6  {
7      siglongjmp(buf, 1);
8  }
9
10 int main()
11 {
12     if (!sigsetjmp(buf, 1)) {
13         Signal(SIGINT, handler);
14         Sio_puts("starting\n");
15     }
16     else
17         Sio_puts("restarting\n");
18
19     while(1) {
20         Sleep(1);
21         Sio_puts("processing...\n");
22     }
23     exit(0); /* Control never reaches here */
24 }
```

code/ecf/restart.c

图 8-44　当用户键入 Ctrl+C 时，使用非本地跳转来重启动它自身的程序

旁注　C++ 和 Java 中的软件异常

C++ 和 Java 提供的异常机制是较高层次的，是 C 语言的 setjmp 和 longjmp 函数的更加结构化的版本。你可以把 try 语句中的 catch 子句看做类似于 setjmp 函数。相似地，throw 语句就类似于 longjmp 函数。