- G2. 保存和恢复 errno。许多 Linux 异步信号安全的函数都会在出错返回时设置 errno。在处理程序中调用这样的函数可能会干扰主程序中其他依赖于 errno 的部分。解决方法是在进入处理程序时把 errno 保存在一个局部变量中，在处理程序返回前恢复它。注意，只有在处理程序要返回时才有此必要。如果处理程序调用 _exit终止该进程，那么就不需要这样做了。
- G3. 阻塞所有的信号，保护对共享全局数据结构的访问。如果处理程序和主程序或其他处理程序共享一个全局数据结构，那么在访问(读或者写)该数据结构时，你的处理程序和主程序应该暂时阻塞所有的信号。这条规则的原因是从主程序访问一个数据结构 d 通常需要一系列的指令，如果指令序列被访问 d 的处理程序中断，那么处理程序可能会发现 d 的状态不一致，得到不可预知的结果。在访问 d 时暂时阻塞信号保证了处理程序不会中断该指令序列。
- G4. 用 volatile 声明全局变量。考虑一个处理程序和一个 main 函数，它们共享一个全局变量 g。处理程序更新 g，main 周期性地读 g。对于一个优化编译器而言，main 中 g 的值看上去从来没有变化过，因此使用缓存在寄存器中 g 的副本来满足对 g 的每次引用是很安全的。如果这样，main 函数可能永远都无法看到处理程序更新过的值。

  可以用 volatile 类型限定符来定义一个变量，告诉编译器不要缓存这个变量。例如：

```
volatile int g;
```

  volatile 限定符强迫编译器每次在代码中引用 g 时，都要从内存中读取 g 的值。一般来说，和其他所有共享数据结构一样，应该暂时阻塞信号，保护每次对全局变量的访问。
- G5. 用 sig_atomic_t 声明标志。在常见的处理程序设计中，处理程序会写全局标志来记录收到了信号。主程序周期性地读这个标志，响应信号，再清除该标志。对于通过这种方式来共享的标志，C 提供一种整型数据类型 sig_atomic_t，对它的读和写保证会是原子的(不可中断的)，因为可以用一条指令来实现它们：

```
volatile sig_atomic_t flag;
```

  因为它们是不可中断的，所以可以安全地读和写 sig_atomic_t 变量，而不需要暂时阻塞信号。注意，这里对原子性的保证只适用于单个的读和写，不适用于像 flag++或 flag=flag+10 这样的更新，它们可能需要多条指令。

要记住我们这里讲述的规则是保守的，也就是说它们不总是严格必需的。例如，如果你知道处理程序绝对不会修改 errno，那么就不需要保存和恢复 errno。或者如果你可以证明 printf 的实例都不会被处理程序中断，那么在处理程序中调用 printf 就是安全的。对共享全局数据结构的访问也是同样。不过，一般来说这种断言很难证明。所以我们建议你采用保守的方法，遵循这些规则，使得处理程序尽可能简单，调用安全函数，保存和恢复 errno，保护对共享数据结构的访问，并使用 volatile 和 sig_atomic_t。

2. 正确的信号处理

信号的一个与直觉不符的方面是未处理的信号是不排队的。因为 pending 位向量中每种类型的信号只对应有一位，所以每种类型最多只能有一个未处理的信号。因此，如果两个类型 k 的信号发送给一个目的进程，而因为目的进程当前正在执行信号 k 的处理程序，所以信号 k 被阻塞了，那么第二个信号就简单地被丢弃了；它不会排队。关键思想是如果存在一个未处理的信号就表明至少有一个信号到达了。