B. 你可以通过用两个不同的 Pthreads 函数调用中的一个替代第 10 行中的 exit 函数来改正这个错误。选哪一个呢？

* 12.18　用图 12-21 中的进度图，将下面的轨迹线分类为安全或者不安全的。

A. H_2，L_2，U_2，H_1，L_1，S_2，U_1，S_1，T_1，T_2

B. H_2，H_1，L_1，U_1，S_1，L_2，T_1，U_2，S_2，T_2

C. H_1，L_1，H_2，L_2，U_2，S_2，U_1，S_1，T_1，T_2

** 12.19　图 12-26 中第一类读者-写者问题的解答给予读者的是有些弱的优先级，因为读者在离开它的临界区时，可能会重启一个正在等待的写者，而不是一个正在等待的读者。推导出一个解答，它给予读者更强的优先级，当写者离开它的临界区的时候，如果有读者正在等待的话，就总是重启一个正在等待的读者。

*** 12.20　考虑读者-写者问题的一个更简单的变种，即最多只有 N 个读者。推导出一个解答，给予读者和写者同等的优先级，即等待中的读者和写者被赋予对资源访问的同等的机会。提示：你可以用一个计数信号量和一个互斥锁来解决这个问题。

*** 12.21　推导出第二类读者-写者问题的一个解答，在此写者的优先级高于读者。

** 12.22　检查一下你对 select 函数的理解，请修改图 12-6 中的服务器，使得它在主服务器的每次迭代中最多只回送一个文本行。

** 12.23　图 12-8 中的事件驱动并发 echo 服务器是有缺陷的，因为一个恶意的客户端能够通过发送部分的文本行，使服务器拒绝为其他客户端服务。编写一个改进的服务器版本，使之能够非阻塞地处理这些部分文本行。

* 12.24　RIO I/O 包中的函数(10.5 节)都是线程安全的。它们也都是可重入函数吗？

* 12.25　在图 12-28 中的预线程化的并发 echo 服务器中，每个线程都调用 echo_cnt 函数(图 12-29)。echo_cnt 是线程安全的吗？它是可重入的吗？为什么是或为什么不是呢？

*** 12.26　用加锁-复制技术来实现 gethostbyname 的一个线程安全而又不可重入的版本，称为 gethostbyname_ts。一个正确的解答是使用由互斥锁保护的 hostent 结构的深层副本。

** 12.27　一些网络编程的教科书建议用以下的方法来读和写套接字：和客户端交互之前，在同一个打开的已连接套接字描述符上，打开两个标准 I/O 流，一个用来读，一个用来写：

```
FILE *fpin, *fpout;

fpin = fdopen(sockfd, "r");
fpout = fdopen(sockfd, "w");
```

当服务器完成和客户端的交互之后，像下面这样关闭两个流：

```
fclose(fpin);
fclose(fpout);
```

然而，如果你试图在基于线程的并发服务器上尝试这种方式，将制造一个致命的竞争条件。请解释。

* 12.28　在图 12-45 中，将两个 V 操作的顺序交换，对程序死锁是否有影响？通过画出四种可能情况的进度图来证明你的答案：

情况 1		情况 2		情况 3		情况 4	
线程 1	线程 2	线程 1	线程 2	线程 1	线程 2	线程 1	线程 2
P(s)	P(s)	P(s)	P(s)	P(s)	P(s)	P(s)	P(s)
P(t)	P(t)	P(t)	P(t)	P(t)	P(t)	P(t)	P(t)
V(s)	V(s)	V(s)	V(t)	V(t)	V(s)	V(t)	V(t)
V(t)	V(t)	V(t)	V(s)	V(s)	V(t)	V(s)	V(s)

* 12.29　下面的程序会死锁吗？为什么会或者为什么不会？