** 12.37　实现一个基于线程的 TINY Web 服务器的并发版本。你的解答应该为每一个新的连接请求创建一个新的线程。使用一个实际的 Web 浏览器来测试你的解答。

** 12.38　实现一个 TINY Web 服务器的并发预线程化的版本。你的解答应该根据当前的负载，动态地增加或减少线程的数目。一个策略是当缓冲区变满时，将线程数量翻倍，而当缓冲区变为空时，将线程数目减半。使用一个实际的 Web 浏览器来测试你的解答。

** 12.39　Web 代理是一个在 Web 服务器和浏览器之间扮演中间角色的程序。浏览器不是直接连接服务器以获取网页，而是与代理连接，代理再将请求转发给服务器。当服务器响应代理时，代理将响应发送给浏览器。为了这个试验，请你编写一个简单的可以过滤和记录请求的 Web 代理：

A. 试验的第一部分中，你要建立以接收请求的代理，分析 HTTP，转发请求给服务器，并且返回结果给浏览器。你的代理将所有请求的 URL 记录在磁盘上一个日志文件中，同时它还要阻塞所有对包含在磁盘上一个过滤文件中的 URL 的请求。

B. 试验的第二部分中，你要升级代理，它通过派生一个独立的线程来处理每一个请求，使得代理能够一次处理多个打开的连接。当你的代理在等待远程服务器响应一个请求使它能服务于一个浏览器时，它应该可以处理来自另一个浏览器未完成的请求。

使用一个实际的 Web 浏览器来检验你的解答。

练习题答案

12.1　当父进程派生子进程时，它得到一个已连接描述符的副本，并将相关文件表中的引用计数从 1 增加到 2。当父进程关闭它的描述符副本时，引用计数就从 2 减少到 1。因为内核不会关闭一个文件，直到文件表中它的引用计数值变为零，所以子进程这边的连接端将保持打开。

12.2　当一个进程因为某种原因终止时，内核将关闭所有打开的描述符。因此，当子进程退出时，它的已连接文件描述符的副本也将被自动关闭。

12.3　回想一下，如果一个从描述符中读一个字节的请求不会阻塞，那么这个描述符就准备好可以读了。假如 EOF 在一个描述符上为真，那么描述符也准备好可读了，因为读操作将立即返回一个零返回码，表示 EOF。因此，键入 Ctrl+D 会导致 `select` 函数返回，准备好的集合中有描述符 0。

12.4　因为变量 `pool.read_set` 既作为输入参数也作为输出参数，所以我们在每一次调用 `select` 之前都重新初始化它。在输入时，它包含读集合。在输出，它包含准备好的集合。

12.5　因为线程运行在同一个进程中，它们都共享相同的描述符表。无论有多少线程使用这个已连接描述符，这个已连接描述符的文件表的引用计数都等于 1。因此，当我们用完它时，一个 `close` 操作就足以释放与这个已连接描述符相关的内存资源了。

12.6　这里的主要的思想是，栈变量是私有的，而全局和静态变量是共享的。诸如 `cnt` 这样的静态变量有点小麻烦，因为共享是限制在它们的函数范围内的——在这个例子中，就是线程例程。

A. 下面就是这张表：

变量实例	被主线程引用?	被对等线程0引用?	被对等线程1引用?
`ptr`	是	是	是
`cnt`	否	是	是
`i.m`	是	否	否
`msgs.m`	是	是	是
`myid.p0`	否	是	否
`myid.p1`	否	否	是

说明：

- `ptr`：一个被主线程写和被对等线程读的全局变量。
- `cnt`：一个静态变量，在内存中只有一个实例，被两个对等线程读和写。
- `i.m`：一个存储在主线程栈中的本地自动变量。虽然它的值被传递给对等线程，但是对等线程也绝不会在栈中引用它，因此它不是共享的。