进程是由内核自动调度的，而且因为它们有各自独立的虚拟地址空间，所以要实现共享数据，必须要有显式的 IPC 机制。事件驱动程序创建它们自己的并发逻辑流，这些逻辑流被模型化为状态机，用 I/O多路复用来显式地调度这些流。因为程序运行在一个单一进程中，所以在流之间共享数据速度很快而且很容易。线程是这些方法的混合。同基于进程的流一样，线程也是由内核自动调度的。同基于 I/O 多路复用的流一样，线程是运行在一个单一进程的上下文中的，因此可以快速而方便地共享数据。

无论哪种并发机制，同步对共享数据的并发访问都是一个困难的问题。提出对信号量的 P 和 V 操作就是为了帮助解决这个问题。信号量操作可以用来提供对共享数据的互斥访问，也对诸如生产者-消费者程序中有限缓冲区和读者-写者系统中的共享对象这样的资源访问进行调度。一个并发预线程化的 echo 服务器提供了信号量使用场景的很好的例子。

并发也引入了其他一些困难的问题。被线程调用的函数必须具有一种称为线程安全的属性。我们定义了四类线程不安全的函数，以及一些将它们变为线程安全的建议。可重入函数是线程安全函数的一个真子集，它不访问任何共享数据。可重入函数通常比不可重入函数更为有效，因为它们不需要任何同步原语。竞争和死锁是并发程序中出现的另一些困难的问题。当程序员错误地假设逻辑流该如何调度时，就会发生竞争。当一个流等待一个永远不会发生的事件时，就会产生死锁。

参考文献说明

信号量操作是 Dijkstra 提出的 [31]。进度图的概念是 Coffman [23]提出的，后来由 Carson 和 Reynolds [16]形式化的。Courtois 等人[25]提出了读者-写者问题。操作系统教科书更详细地描述了经典的同步问题，例如哲学家进餐问题、打瞌睡的理发师问题和吸烟者问题 [102, 106, 113]。Butenhof 的书 [15]对 Posix 线程接口有全面的描述。Birrell [7]的论文对线程编程以及线程编程中容易遇到的问题做了很好的介绍。Reinders 的书[90]描述了 C/C++ 库，简化了线程化程序的设计和实现。有一些课本讲述了多核系统上并行编程的基础知识 [47, 71]。Pugh 描述了 Java 线程通过内存进行交互的方式的缺陷，并提出了替代的内存模型 [88]。Gustafson 提出了替代强扩展的弱扩展加速模型 [43]。

家庭作业

* 12.16　编写 hello.c(图 12-13)的一个版本，它创建和回收 n 个可结合的对等线程，其中 n 是一个命令行参数。

* 12.17　A. 图 12-46 中的程序有一个 bug。要求线程睡眠一秒钟，然后输出一个字符串。然而，当在我们的系统上运行它时，却没有任何输出。为什么？

code/conc/hellobug.c

```
1   /* WARNING: This code is buggy! */
2   #include "csapp.h"
3   void *thread(void *vargp);
4
5   int main()
6   {
7       pthread_t tid;
8
9       Pthread_create(&tid, NULL, thread, NULL);
10      exit(0);
11  }
12
13  /* Thread routine */
14  void *thread(void *vargp)
15  {
16      Sleep(1);
17      printf("Hello, world!\n");
18      return NULL;
19  }
```

code/conc/hellobug.c

图 12-46　练习题 12.17 的有 bug 的程序