并发程序：使用应用级并发的程序称为并发程序 (那还有什么级并发程序)
基本构造并发程序的方法：

- 进程
- I/O多路复用 (不懂)
- 线程：是运行在单一进程上下文中的逻辑流，有内核进行调度，并共享同一虚拟地址空间。

12.1 基于进程的并发编程

使用fork()，创建子进程实现。

12.1.1 基于进程的并发服务器

12.1.2 关于进程的优劣

进程有独立的地址空间既是优点也是缺点。
优点是其他进程不可能覆盖该进程的存储器空间。
缺点是进程间的通信变得困难，需要使用显式的IPC(进程间通信)机制。
IPC机制有：管道通信，共享内存，消息队列，信号量。

12.2 基于I/O多路复用的并发编程

12.3 基于线程的并发编程

线程就是运行在进程上下文中的逻辑流。

每个线程都有它自己的线程上下文，包括一个唯一的整数线程ID、栈、栈指针、程序计数器、通用目的寄存器和条件码。
所有运行在一个进程里的线程共享该进程的整个虚拟地址空间。

12.3.1 线程执行模型

线程与进程的区别：

- 线程的切换比进程切换要快得多。
- 线程不像进程那样，不是按照严格得父子层次来组织得。和一个进程相关的线程组成一个对等(线程)池，。
  一个线程可以杀死它的任何对等线程，或者等待它的任意对等线程终止。

13.3.6 分离线程

默认情况下，线程被创建成可结合的。一个可结合的线程能够被其他线程回收其资源和杀死。在被其他线程回收之前，它的存储器资源是没有被释放的。

每个可结合线程都应该要么被其他线程显式地收回，要么通过调用pthread_detach函数被分离。

12.4 多线程程序中的共享变量

线程很有吸引力的一个方面就是多个线程很容易共享相同的程序变量。

12.4.1 线程存储器模型

线程中的通用目的寄存器是不共享的。

12.4.2 将变量映射到存储器

- 全局变量：在运行时，虚拟存储器的读写区域只包含每个全局变量的一个实例，任何线程都可以引用。
- 本地自动变量：定义在函数内部，每个线程的栈都包含它自己的所有本地自动变量的实例。
- 本地静态变量：函数内部并有static的变量。和全局变量一样，虚拟存储器的读/写区域只包含在程序中声明的每个本地静态变量的一个实例。

12.4.3 共享变量

说一个变量是共享的，当且仅当它的一个实例被一个以上的线程引用。

12.5 用信号量同步线程

12.5.2 信号量

信号量是具有非负整数值的全局变量，只能由两种特殊的操作来处理，这两种操作称为P和V：

- P(s)：如果s非零，那么P将s减1，并且立即返回。如果s为零，则挂起这个线程。直到s变为非零。
- V(s)：V操作将s加1。若在P操作有线程阻塞，会唤醒线程，完成它的P操作。
  操作信号量的函数：
  ""
  int sem_init(sem_t *sem, 0, unsigned int value)
  int sem_wait(sem_t *s);
  int sem_post(sem_t *s);
  ""

12.5.3 使用信号量实现互斥

使用的信号量叫做二元信号量，其值只能为0或者1。也称为互斥锁。

- 加锁
- 解锁

12.5.4 利用信号量来调度共享资源

- 生产者和消费者
- 读锁和写锁

12.7.5 死锁

死锁是指一组线程被阻塞，等待一个永远也不可能为真的条件。