

# 面试总结-----操作系统

## 1. 操作系统特点

- 并发性、共享性、虚拟性、不确定性。

## 2. 什么是进程

1. 进程是指在系统中正在运行的一个应用程序，程序一旦运行就是进程；
2. 进程可以认为是程序执行的一个实例，进程是系统进行资源分配的最小单位，且每个进程拥有独立的地址空间；
3. 一个进程无法直接访问另一个进程的变量和数据结构，如果希望一个进程去访问另一个进程的资源，需要使用进程间的通信，比如：管道、消息队列等
4. 线程是进程的一个实体，是进程的一条执行路径；比进程更小的独立运行的基本单位，线程也被称为轻量级进程，一个程序至少有一个进程，一个进程至少有一个线程；

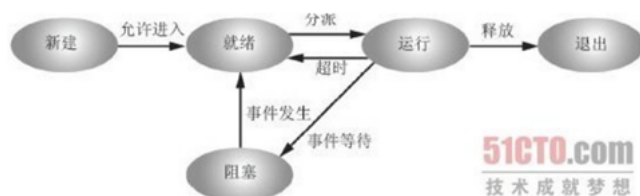
## 3. 进程

1. 进程是程序的一次执行，该程序可以与其他程序并发执行；
2. 进程有运行、阻塞、就绪三个基本状态；
3. 进程调度算法：先来先服务调度算法、短作业优先调度算法、非抢占式优先级调度算法、抢占式优先级调度算法、高响应比优先调度算法、时间片轮转法调度算法；

## 4. 进程与线程的区别

1. 同一进程的线程共享本进程的地址空间，而进程之间则是独立的地址空间；
2. 同一进程内的线程共享本进程的资源，但是进程之间的资源是独立的；
3. 一个进程崩溃后，在保护模式下不会对其他进程产生影响，但是一个线程崩溃整个进程崩溃，所以多进程比多线程健壮；
4. 进程切换，消耗的资源大。所以涉及到频繁的切换，使用线程要好于进程；
5. 两者均可并发执行；
6. 每个独立的进程有一个程序的入口、程序出口。但是线程不能独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

## 5. 进程状态转换图



1. 新状态：进程已经创建
2. 就绪态：进程做好了准备，准备执行，等待分配处理机
3. 执行态：该进程正在执行；
4. 阻塞态：等待某事件发生才能执行，如等待I/O完成；
5. 终止状态

## 6. 进程的创建过程？需要哪些函数？需要哪些数据结构？

1. fork函数创造的子进程是父进程的完整副本，复制了父亲进程的资源，包括内存的内容task\_struct内容；
2. vfork创建的子进程与父进程共享数据段，而且由vfork创建的子进程将先于父进程运行；
3. linux上创建线程一般使用的是pthread库，实际上linux也给我们提供了创建线程的系统调用，就是clone；

## 7. 子进程和父进程怎么通信？

1. 在Linux系统中实现父子进程的通信可以采用pipe()和fork()函数进行实现；
2. 对于父子进程，在程序运行时首先进入的是父进程，其次是子进程，在此我个人认为，在创建父子进程的时候程序是先运行创建的父进程，其次在复制父进程创建子进程。fork()函数主要是以父进程为蓝本复制一个进程，其ID号和父进程的ID号不同。对于结果fork出来的子进程的父进程ID号是执行fork()函数的进程的ID号。

- 管道：是指用于连接一个读进程和一个写进程，以实现它们之间通信的共享文件，又称pipe文件。
- 写进程在管道的尾端写入数据，读进程在管道的首端读出数据。

## 8. 进程和作业的区别？

- 进程是程序的一次动态执行，属于动态概念；
- 一个进程可以执行一个或几个程序，同一个程序可由几个进程执行；
- 程序可以作为一种软件资源长期保留，而进程是程序的一次执行；
- 进程具有并发性，能与其他进程并发执行；
- 进程是一个独立的运行单位；

## 9. 死锁是什么？必要条件？如何解决？

- 所谓死锁，是指多个进程循环等待它方占有的资源而无限期地僵持下去的局面。很显然，如果没有外力的作用，那么死锁涉及到的各个进程都将永远处于封锁状态。当两个或两个以上的进程同时对多个互斥资源提出使用要求时，有可能导致死锁。
  - 互斥条件。即某个资源在一段时间内只能由一个进程占有，不能同时被两个或两个以上的进程占有。这种独占资源如CD-ROM驱动器，打印机等等，必须在占有该资源的进程主动释放它之后，其它进程才能占有该资源。这是由资源本身的属性所决定的。如独木桥就是一种独占资源，两方的人不能同时过桥。
  - 不可抢占条件。进程所获得的资源在未使用完毕之前，资源申请者不能强行地从资源占有者手中夺取资源，而只能由该资源的占有者进程自行释放。如过独木桥的人不能强迫对方后退，也不能非法地将对方推下桥，必须是桥上的人自己过桥后空出桥面（即主动释放占有资源），对方的人才能过桥。
  - 占有且申请条件。进程至少已经占有一个资源，但又申请新的资源；由于该资源已被另外进程占有，此时该进程阻塞；但是，它在等待新资源之时，仍继续占用已占有的资源。还以过独木桥为例，甲乙两人在桥上相遇。甲走过一段桥面（即占有了一些资源），还需要走其余的桥面（申请新的资源），但那部分桥面被乙占有（乙走过一段桥面）。甲过不去，前进不能，又不后退；乙也处于同样的状况。
  - 循环等待条件。存在一个进程等待序列{P1, P2, ..., Pn}，其中P1等待P2所占有的某一资源，P2等待P3所占有的某一资源，.....，而Pn等待P1所占有的某一资源，形成一个进程循环等待环。就像前面的过独木桥问题，甲等待乙占有的桥面，而乙又等待甲占有的桥面，从而彼此循环等待。
- 死锁的预防是保证系统不进入死锁状态的一种策略。它的基本思想是要求进程申请资源时遵循某种协议，从而打破产生死锁的四个必要条件中的一个或几个，保证系统不会进入死锁状态。
  - 打破互斥条件。即允许进程同时访问某些资源。但是，有的资源是不允许被同时访问的，像打印机等等，这是由资源本身的属性所决定的。所以，这种方法并无实用价值。
  - 打破不可抢占条件。即允许进程强行从占有者那里夺取某些资源。就是说，当一个进程已占有了某些资源，它又申请新的资源，但不能立即被满足时，它必须释放所占有的全部资源，以后再重新申请。它所释放的资源可以分配给其它进程。这就相当于该进程占有的资源被隐蔽地强占了。这种预防死锁的方法实现起来困难，会降低系统性能。
  - 打破占有且申请条件。可以实行资源预先分配策略。即进程在运行前一次性地向系统申请它所需要的全部资源。如果某个进程所需的全部资源得不到满足，则不分配任何资源，此进程暂不运行。只有当系统能够满足当前进程的全部资源需求时，才一次性地将所申请的资源全部分配给该进程。由于运行的进程已占有了它所需的全部资源，所以不会发生占有资源又申请资源的现象，因此不会发生死锁。
  - 打破循环等待条件，实行资源有序分配策略。采用这种策略，即把资源事先分类编号，按号分配，使进程在申请，占用资源时不会形成环路。所有进程对资源的请求必须严格按资源序号递增的顺序提出。进程占用了小号资源，才能申请大号资源，就不会产生环路，从而预防了死锁。

- 死锁避免：银行家算法

## 10. 鸵鸟策略

- 假设的前提是，这样的问题出现的概率很低。比如，在操作系统中，为应对死锁问题，可以采用这样的一种办法。当系统发生死锁时不会对用户造成多大影响，或系统很少发生死锁的场合采用允许死锁发生的鸵鸟算法，这样一来可能开销比不允许发生死锁及检测和解除死锁的小。如果死锁很长时间才发生一次，而系统每周都会因硬件故障、编译器错误或操作系统错误而崩溃一次，那么大多数工程师不会以性能损失或者易用性损失的代价来设计较为复杂的死锁解决策略，来消除死锁。
- 鸵鸟策略的实质：出现死锁的概率很小，并且出现之后处理死锁会花费很大的代价，还不如不做处理，OS中这种置之不理的策略称之为鸵鸟策略（也叫鸵鸟算法）。

## 11. 银行家算法

- 在避免死锁的方法中，所施加的限制条件较弱，有可能获得令人满意的系统性能。在该方法中把系统的状态分为安全状态和不安全状态，只要能使系统始终都处于安全状态，便可以避免发生死锁。
- 银行家算法的基本思想是分配资源之前，判断系统是否是安全的；若是，才分配。它是最具有代表性的避免死锁的算法。

## 12. 进程间通信方式有几种，他们之间的区别是什么？

1. 管道(pipe)

管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

有名管道 (namedpipe)

有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

信号量(semaphore)

信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

消息队列(messagequeue)

消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

信号 (sinal)

信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

共享内存(shared memory)

共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量，配合使用，来实现进程间的同步和通信。

套接字(socket)

套接口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同设备及其间的进程通信。

13. 页和段的区别？

- 1. 页是信息的物理单位，分页是由于系统管理的需要。段是信息的逻辑单位，分段是为了满足用户的要求。
- 2. 页的大小固定且由系统决定，段的长度不固定，决定于用户所编写的程序，通常由编译程序在对源程序紧进行编译时，根据信息的性质来划分。
- 3. 分页的作业的地址空间是一维的，程序员只需要利用一个记忆符，即可表示一个地址。分段的作业地址空间则是二维的，程序员在标识一个地址时，既要给出段名，又需要给出段的地址值。

14. 线程和进程的区别？线程共享的资源是什么？

- 1. 一个程序至少有一个进程，一个进程至少有一个线程
- 2. 线程的划分尺度小于进程，使得多线程程序的并发性高
- 3. 进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率
- 4. 每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制
- 5. 多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。
- 6. 一个进程中的所有线程共享该进程的地址空间，但它们有各自独立的（/私有的）栈(stack)，Windows线程的缺省堆栈大小为1M。堆(heap)的分配与栈有所不同，一般是一个进程有一个C运行时堆，这个堆为本进程中所有线程共享，windows进程还有所谓进程默认堆，用户也可以创建自己的堆。

线程共享资源	线程私有资源
地址空间	程序计数器
全局变量	寄存器
打开的文件	栈
子进程	状态字
周轮	
信号及信号服务程序	
记账信息	

- 线程私有：线程栈，寄存器，程序寄存器
- 共享：堆，地址空间，全局变量，静态变量
- 进程私有：地址空间，堆，全局变量，栈，寄存器
- 共享：代码段，公共数据，进程目录，进程

15. 进程、线程、协程的区别

- 1. 进程：进程（Process）是计算机中的程序关于某数据集上的一次运行活动，是系统进行资源分配和调度的基本单位，是操作系统结构的基础。
- 进程的概念主要有两点：第一，进程是一个实体。每一个进程都有它自己的地址空间，一般情况下，包括文本区域（text region）、数据区域（data region）和堆栈（stack region）。

- 第二，进程是一个“执行中的程序”。程序是一个没有生命的实体，只有[处理器](#)赋予程序生命时（操作系统执行之），它才能成为一个活动的实体，我们称其为[进程](#)。

1. 线程：**线程是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源。**线程间通信主要通过共享内存，上下文切换很快，资源开销较少，但相比进程不够稳定容易丢失数据。

2. **协程是一种用户态的轻量级线程**，协程的调度完全由**用户控制**。协程拥有自己的寄存器上下文和栈。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈，直接操作栈则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快。

## • 进程、线程共同点

它们都能提高程序的并发度，提高程序运行效率和响应时间。线程和进程在使用上各有优缺点。线程执行开销比较小，但不利于资源的管理和保护，而进程相反。同时，线程适合在SMP机器上运行，而进程可以跨机器迁移。

## • 进程、线程不同点

多进程中每个进程有自己的地址空间，线程则共享地址空间。

所有其他区别都是因为这个区别产生的。比如说：

- 1) 地址空间:线程是进程内的一个执行单元，进程内至少有一个线程，它们共享进程的地址空间，而进程有自己独立的地址空间
- 2) 资源拥有:进程是资源分配和拥有的单位,同一个进程内的线程共享进程的资源
- 3) 线程是处理器调度的基本单位,但进程不是
- 4) 二者均可并发执行
- 5) 每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口，但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制
1. 速度。线程产生的速度快，通讯快，切换快，因为他们处于同一地址空间。
2. 线程的资源利用率好。
3. 线程使用公共变量或者内存的时候需要同步机制，但进程不用。

而他们通信方式的差异也仍然是由于这个根本原因造成的。

## 16. 线程比进程具有哪些优势？

1. 线程在程序中是独立的，并发的执行流，但是，[进程中的线程之间的隔离程度要小](#)；
2. 线程比进程更具有更高的性能，这是由于同一个进程中的线程都有共性：[多个线程将共享](#)同一个进程虚拟空间；
3. 当操作系统[创建一个进程时](#)，必须为进程分配独立的内存空间，并分配大量相关资源；

## 17. 什么时候用多进程？什么时候用多线程？

1. 需要[频繁创建销毁](#)的优先用线程；
2. 需要进行[大量计算](#)的优先使用线程；
3. 强相关的处理用线程，弱相关的处理用进程；
4. 可能要扩展到多机分布的用进程，[多核分布](#)的用线程；

## 18. 协程是什么？

1. 是一种[比线程更加轻量级](#)的存在。正如一个进程可以拥有多个线程一样，[一个线程可以拥有多个协程](#)；协程不是被操作系统内核管理，而[完全是由程序所控制](#)。
2. 协程的开销远远小于线程；
3. 协程[拥有自己寄存器上下文和栈](#)。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切换回来的时候，恢复先前保存的寄存器上下文和栈。
4. [每个协程表示一个执行单元](#)，有自己的本地数据，与其他协程共享全局数据和其他资源。
5. 跨平台、跨体系架构、无需线程上下文切换的开销、方便切换控制流，简化编程模型；
6. 协程又称为微线程，[协程的完成主要靠yield关键字](#)，协程执行过程中，在子程序内部可中断，然后转而执行别的子程序，在适当的时候再返回来接着执行；
7. 协程极高的执行效率，和多线程相比，线程数量越多，协程的性能优势就越明显；
8. 不需要多线程的锁机制；

## 19. 用户态到内核态的转化原理？



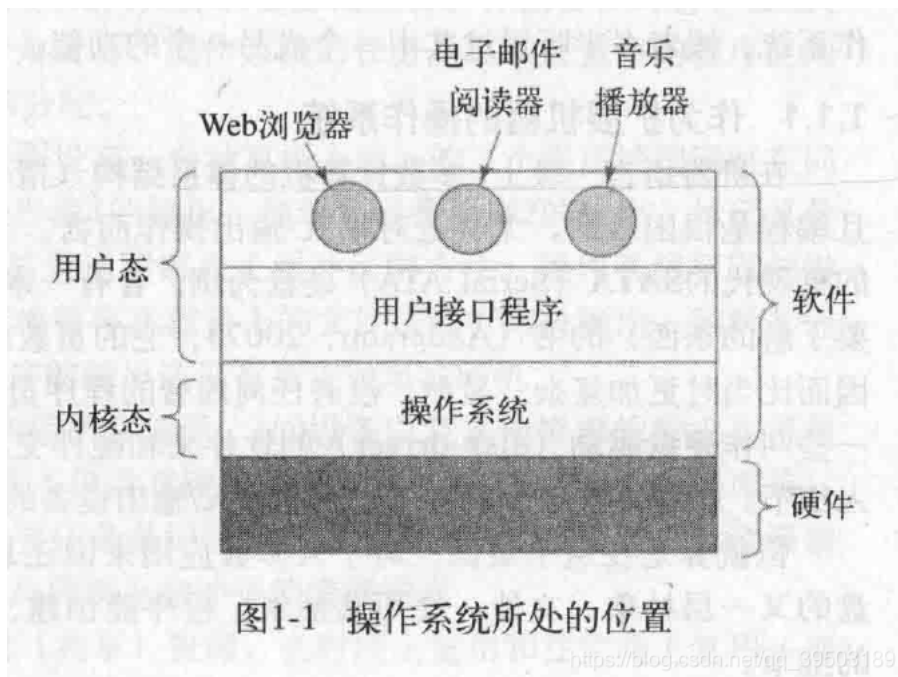


图1-1 操作系统所处的位置

[https://blog.csdn.net/qq\\_39503189](https://blog.csdn.net/qq_39503189)

## • 系统调用

1. 这是用户进程主动要求切换到内核态的一种方式，用户进程通过系统调用申请操作系统提供的服务程序完成工作。而系统调用的机制其核心还是使用了操作系统为用户特
3. 别开放的一个中断来实现，例如Linux的ine 80h中断。

## • 异常

当CPU在执行运行在用户态的程序时，发现了某些事件不可知的异常，这是会触发由当前运行进程切换到处理此。异常的内核相关程序中，也就到了内核态，比如缺页异常。

## • 外围设备的中断

当外围设备完成用户请求的操作之后，会向CPU发出相应的中断信号，这时CPU会暂停执行下一条将要执行的指令，转而去执行中断信号的处理程序，如果先执行的指令是用户态下的程序，那么这个转换的过程自然也就发生了有用户态到内核态的切换。比如硬盘读写操作完成，系统会切换到硬盘读写的中断处理程序中执行后续操作等。

## 2) 切换操作

从出发方式看，可以在认为存在前述3种不同的类型，但是从最终实际完成由用户态到内核态的切换操作上来说，涉及的关键步骤是完全一样的，没有任何区别，都相当于执行了一个中断响应的过程，因为系统调用实际上最终是中断机制实现的，而异常和中断处理机制基本上是一样的，用户态切换到内核态的步骤主要包括：

- 1、从当前进程的描述符中提取其内核栈的ss0及esp0信息。
- 2、使用ss0和esp0指向的内核栈将当前进程的cs,eip, eflags, ss,esp信息保存起来，这个过程也完成了由用户栈找到内核栈的切换过程，同时保存了被暂停执行的程序的下一条指令。
- 3、将先前由中断向量检索得到的中断处理程序的cs, eip信息装入相应的寄存器，开始执行中断处理程序，这时就转到了内核态的程序执行了。

## 20. 系统中断是什么，用户态和内核态的区别

1. 内核态与用户态是操作系统的两种运行级别,当**程序运行在3级特权级上时**，就可以称之为运行在**用户态**，因为这是最低特权级，是普通的用户进程运行的特权级，大部分用户直接面对的程序都是运行在用户态；反之，**当程序运行在0级特权级上时**，就可以称之为运行在**内核态**。运行在**用户态下的程序不能直接访问操作系统内核数据结构和程序**。当我们在系统中执行一个程序时，大部分时间是运行在用户态下的，在其需要操作系统帮助完成某些它没有权力和能力完成的工作时就会切换到内核态。
2. 这两种状态的主要差别是：处于**用户态**执行时，进程所能访问的**内存空间**和**对象**受到限制，其所处于占有的**处理机是可被抢占的**；而处于**核心态**执行中的进程，则**能访问所有的内存空间和对象**，且所占有的**处理机是不允许被抢占的**。

## 21. 虚拟内存？使用虚拟内存的优点？什么是虚拟地址空间？

1. 虚拟内存，**虚拟内存是一种内存管理技术**，它会使程序自己认为自己拥有一块很大且连续的内存，然而，这个程序在内存中不是连续的，并且有些还会在磁盘上，在需要时进行数据交换；
2. 优点：可以**弥补物理内存大小的不足**；一定程度的提高反应速度；减少对物理内存的读取从而保护内存延长内存使用寿命；
3. 缺点：**占用一定的物理硬盘空间**；加大了对硬盘的读写；设置不当会影响整机稳定性与速度。
4. 虚拟地址空间是对于一个**单一进程**的概念，这个进程看到的将是地址从**0000**开始的整个内存空间。虚拟存储器是一个抽象概念，它为每一个进程提供了一个假象，好像**每一个进程都在独占的使用主存**。每个进程看到的存储器都是一致的，称为虚拟地址空间。从最低的地址看起：**程序代码和数据，堆，共享库，栈，内核虚拟存储器**。大多数计算机的字长都是32位，这就限制了虚拟地址空间为4GB。

## 22. 线程安全？如何实现？

1. 如果你的代码所在的**进程中有多个线程在同时运行**，而这些**线程可能会同时运行这段代码**。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，就是线程安全的。
2. 线程安全问题都是由**全局变量及静态变量**引起的。
3. 若每个线程中对全局变量、静态变量**只有读操作**，而无写操作，一般来说，这个全局变量是线程安全的；若有**多个线程同时执行写操作**，一般都需要考虑**线程同步**，否则的话就可能影响线程安全。

• 对于线程不安全的对象我们可以通过如下方法来实现线程安全：

1. **加锁** 利用Synchronized或者ReentrantLock来对不安全对象进行加锁，来实现线程执行的串行化，从而保证多线程同时操作对象的安全性，一个是语法层面的互斥锁，一个是API层面的互斥锁。
2. **非阻塞同步**来实现线程安全。原理就是：通俗点讲，就是先进性操作，如果没有其他线程争用共享数据，那操作就成功了；如果共享数据有争用，产生冲突，那就再采取其他措施(最常见的措施就是不断地重试，知道成功为止)。这种方法需要硬件的支持，因为我们需要操作和冲突检测这两个步骤具备原子性。通常这种指令包括CAS,SC,FAI,TAS等。
3. **线程本地化**，一种无同步的方案，就是利用ThreadLocal来为**每一个线程创建一个共享变量的副本**来（副本之间是无关的）避免几个线程同时操作一个对象时发生线程安全问题。

## 23. linux文件系统

• 层次分析

1. 用户层，日常使用的各种程序，需要的接口主要是文件的创建、删除、读、写、关闭等；
2. VFS层，文件相关的操作都有对应的System Call函数接口，接口调用VFS对应的函数；
3. 文件系统层，用户的操作通过VFS转到各种文件系统。文件系统把文件读写命令转化为对磁盘LBA的操作，起了一个翻译和磁盘管理的工作；
4. 缓存层；
5. 块设备层，块设备接口Block Device是用来访问磁盘LBA的层级，读写命令组合之后插入到命令队列，磁盘的驱动从队列读命令执行；
6. 磁盘驱动层；
7. 磁盘物理层；

• 读取文件过程

1. 根据文件所在目录的inode信息，找到目录文件对应数据块；
2. 根据文件名从数据块中找到对应的inode节点信息；
3. 从文件inode节点信息中找到文件内容所在数据块块号；
4. 读取数据块内容

## 24. 常见的IO模型，五种？异步IO应用场景？有什么缺点？

1. **同步**

• 就是在发出一个功能调用时，在**没有得到结果之前，该调用就不返回**。也就是**必须一件一件事做**，等前一件做完了才能做下一件事。就是我调用一个功能，该功能没有结束前，我死等结果。

1. **异步**

• 当一个**异步过程调用发出后**，调用者不能立刻得到结果。实际处理这个调用的部件在完成后，通过**状态、通知和回调来通知调用者**。就是我调用一个功能，不需要知道该功能结果，该功能有结果后通知我（回调通知）

1. **阻塞**

- 阻塞调用是指调用结果返回之前，当前线程会被挂起（线程进入非可执行状态，在这个状态下，cpu不会给线程分配时间片，即线程暂停运行）。函数只有在得到结果之后才会返回。对于同步调用来说，很多时候当前线程还是激活的，只是从逻辑上当前函数没有返回而已。就是调用我（函数），我（函数）没有接收完数据或者没有得到结果之前，我不会返回。

非阻塞指在不能立刻得到结果之前，该函数不会阻塞当前线程，而会立刻返回。就是调用我（函数），我（函数）立即返回，通过select通知调用

## 25. Linux是如何避免内存碎片的

1. 在固定式分区分配中, 为将一个用户作业装入内存, 内存分配程序从系统分区表中找出一个能满足作业要求的空闲分区分配给作业, 由于一个作业的大小并不一定与分区大小相等, 因此, 分区中有一部分存储空间浪费掉了. 由此可知, 固定式分区分配中存在内存碎片.
2. 在可变式分区分配中, 为把一个作业装入内存, 应按照一定的分配算法从系统中找出一个能满足作业需求的空闲分区分配给作业, 如果这个空闲分区的容量比作业申请的空间容量要大, 则将该分区一分为二, 一部分分配给作业, 剩下的部分仍然留作系统的空闲分区. 由此可知, 可变式分区分配中存在内存碎片.
3. 伙伴系统
4. 据可移动性组织页避免内存碎片

## 26. 计一个线程池，内存池

1. 为什么需要线程池

- 大多数的网络服务器，包括Web服务器都具有一个特点，就是单位时间内必须处理数目巨大的连接请求，但是处理时间却是比较短的。在传统的多线程服务器模型中是这样实现的：一旦有个请求到达，就创建一个新的线程，由该线程执行任务，任务执行完毕之后，线程就退出。这就是“即时创建，即时销毁”的策略。尽管与创建进程相比，创建线程的时间已经大大的缩短，但是如果提交给线程的任务是执行时间较短，而且执行次数非常频繁，那么服务器就将处于一个不停的创建线程和销毁线程的状态。这笔开销是不可忽略的，尤其是线程执行的时间非常非常短的情况。

1. 线程池原理

- 在应用程序启动之后，就马上创建一定数量的线程，放入空闲的队列中。这些线程都是处于阻塞状态，这些线程只占一点内存，不占用CPU。当任务到来后，线程池将选择一个空闲的线程，将任务传入此线程中运行。当所有的线程都处在处理任务的时候，线程池将自动创建一定的数量的新线程，用于处理更多的任务。执行任务完成之后线程并不退出，而是继续在线程池中等待下一次任务。当大部分线程处于阻塞状态时，线程池将自动销毁一部分的线程，回收系统资源。

1. 线程池的作用

- 需要大量的线程来完成任务，且完成任务的时间比较短；对性能要求苛刻的应用；对性能要求苛刻的应用




1. 内存池的原理

- 在软件开发中，有些对象使用非常频繁，那么我们可以预先在堆中实例化一些对象，我们把维护这些对象的结构叫“内存池”。在需要用的时候，直接从内存池中拿，而不用从新实例化，在要销毁的时候，不是直接free/delete，而是返还给内存池。把那些常用的对象存在内存池中，就不用频繁的分配/回收内存，可以相对减少内存碎片，更重要的是实例化这样的对象更快，回收也更快。当内存池中的对象不够用的时候就扩容。

1. 内存池的优缺点

- 内存池对象不是线程安全的，在多线程编程中，创建一个对象时必须加锁。

相关推荐

关于我们 招贤纳士 广告服务 开发助手  400-660-0108  kefu@csdn.net  在线客服 工作时间 8:30-22:00

公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心  
网络110报警服务 中国互联网举报中心 家长监护 Chrome商店下载 ©1999-2021北京创新乐知网络技术有限公司 版权与免责声明 版权申诉  
出版物许可证 营业执照