1 进程与线程的区别

进程是程序运行的实例，一个程序至少有一个进程，进程是系统进行资源分配的基本单位，一个进程至少包含一个线程，进程有自己独立的地址空间，这些地址空间被所有线程所共享。而线程可以看成是轻量级的进程，它是进程中负责程序执行的执行单元，也是 CPU 调度和分派的基本单位。线程自己不拥有系统资源（调度所付出的开销小，上下文切换快），只拥有一点在运行时必不可少的资源，但它可与同属一个进程的其他线程共享进程所拥有的全部资源。线程中包含线程 ID、程序计数器 PC、寄存器、栈及状态信息（准备|就绪|运行|阻塞|结束）

2 进程调度算法

先来先服务(FCFS)调度算法

短作业优先(SJF)调度算法

优先级调度算法

高响应比优先调度算法

时间片轮转调度算法

3 进程间的通信

（1）信号：是在软件层次上对中断机制的一种模拟。在原理上，一个进程收到一个信号与处理器收到一个中断请求可以说是一样的，信号是进程间通信机制中唯一的异步通信机制

（2）信号量：可以说是一个计数器，用来处理进程或线程同步的问题，特别是对临界资源的访问同步问题。信号量的值仅能由 PV（P 表示通过 V 表示释放）操作来改变

（3）消息队列：是存放在内核中的消息链表，每个消息队列由消息队列标识符标识，与管道不同的是，消息队列存放在内核中，只有在内核重启时才能删除一个消息队列，内核重启也就是系统重启，同样消息队列的大小也是受限制的

（4）共享内存：共享内存就是分配一块能被其他进程访问的内存。共享内存可以说是最有用的进程间通信方式，也是最快的 IPC 形式

（5）管道：管道传递数据是单向性的，只能从一方流向另一方，只用于有亲缘关系的进程间的通信，亲缘关系也就是父子进程或兄弟进程

（6）套接字

共享内存方式的优缺点 速度快 但需要同步

线程与进程间通信方式

线程属于这个进程，直接访问这个进程的地址空间就行了

线程不属于这个进程——进程与进程之间的通信

4 怎样避免死锁

死锁是指多个进程因竞争资源而造成的一种僵局（互相等待） 两车过桥

死锁产生的原因：系统资源的竞争、进程运行推进顺序不合理

产生死锁的四个必要条件：互斥条件（指被争夺的资源）、请求与保持条件（进程吃着碗里的看着锅里的）、不可剥夺条件（进程占有的资源不可被剥夺）、循环等待条件

死锁的处理：忽略死锁（处理的代价高于忽略）鸵鸟算法、剥夺资源法、进程回退法、杀死进程、重启系统

如何预防死锁：银行家算法（动态检测资源分配，确保循环等待条件永远不成立）、破坏条件等

如何解决死锁

（1）进程在申请资源时，一次性得到所需要的所有资源，若无法满足则不能执行

（2）进程在申请新的资源时，释放已占有的资源，后面若还需要它们，则需要重新申请

（3）对系统中的资源顺序编号，规定进程只能依次申请资源

5 操作系统的段存储，页存储，段页存储的区别

分页存储:

用户程序的地址空间被划分成若干固定大小的区域，称为“页”，相应地，内存空间分成若干个物理块，页和块的大小相等。可将用户程序的任一页放在内存的任一块中，实现了离散分配。

分段存储:

页面是主存物理空间中划分出来的等长的固定区域。分页方式的优点是页长固定，因而便于构造页表、易于管理，且不存在外碎片。但分页方式的缺点是页长与程序的逻辑大小不相关。例如，某个时刻一个子程序可能有一部分在主存中，另一部分则在辅存中。这不利于编程时的独立性，并给换入换出处理、存储保护和存储共享等操作造成麻烦。

另一种划分可寻址的存储空间的方法称为分段。段是按照程序的自然分界划分的长度可以动态改变的区域。通常，程序员把子程序、操作数和常数等不同类型的数据划分到不同的段中，并且每个程序可以有多个相同类型的段。

段表本身也是一个段，可以存在辅存中，但一般是驻留在主存中。

将用户程序地址空间分成若干个大小不等的段，每段可以定义一组相对完整的逻辑信息。存储分配时，以段为单位，段与段在内存中可以不相邻接，也实现了离散分配。

段页式存储:

段页式存储组织是分段式和分页式结合的存储组织方法，这样可充分利用分段管理和分页管理的优点。

(1) 用分段方法来分配和管理虚拟存储器。程序的地址空间按逻辑单位分成基本独立的段，而每一段有自己的段名，再把每段分成固定大小的若干页。

(2) 用分页方法来分配和管理实存。即把整个主存分成与上述页大小相等的存储块，可装入作业的任何一页。程序对内存的调入或调出是按页进行的。但它又可按段实现共享和保护。

6 虚拟内存原理 内存分页

https://blog.csdn.net/dong_007_007/article/details/16339883

https://blog.csdn.net/chluknight/article/details/6689323

7.内存池实现

C/C++下内存管理是让几乎每一个程序员头疼的问题，分配足够的内存、追踪内存的分配、在不需要的时候释放内存——这个任务相当复杂。而直接使用系统调用 malloc/free、new/delete 进行内存分配和释放，有以下弊端：

调用 malloc/new,系统需要根据“最先匹配”、“最优匹配”或其他算法在内存空闲块表中查找一块空闲内存，调用 free/delete,系统可能需要合并空闲内存块，这些会产生额外开销

频繁使用时会产生大量内存碎片，从而降低程序运行效率

容易造成内存泄漏

内存池则是在真正使用内存之前，先申请分配一定数量的、大小相等(一般情况下)的内存块留作备用。当有新的内存需求时，就从内存池中分出一部分内存块，若内存块不够再继续申请新的内存。这样做的一个显著优点是，使得内存分配效率得到提升。

8 Linux 管道机制

在 Linux 中，管道是一种使用非常频繁的通信机制。从本质上说，管道也是一种文件，但它又和一般的文件有所不同，管道可以克服使用文件进行通信的两个问题，具体表现为：

1）限制管道的大小。实际上，管道是一个固定大小的缓冲区。在 Linux 中，该缓冲区的大小为 1 页，即 4K 字节，使得它的大小不像文件那样不加检验地增长。使用单个固定缓冲区也会带来问题，比如在写管道时可能变满，当这种情况发生时，随后对管道的 write()调用将默认地被阻塞，等待某些数据被读取，以便腾出足够的空间供 write()调用写。

2）读取进程也可能工作得比写进程快。当所有当前进程数据已被读取时，管道变空。当这种情况发生时，一个随后的 read()调用将默认地被阻塞，等待某些数据被写入，这解决了 read()调用返回文件结束的问题。

注意：从管道读数据是一次性操作，数据一旦被读，它就从管道中被抛弃，释放空间以便写更多的数据。

9 linux 中硬链接和软链接的区别

https://www.cnblogs.com/xuxiuxiu/p/6294152.html#top

若一个 inode 号对应多个文件名，则称这些文件为硬链接。换言之，硬链接就是同一个文件使用了多个别名（见 图 2.hard link 就是 file 的一个别名，他们有共同的 inode）。硬链接可由命令 link 或 ln 创建。如下是对文件 oldfile 创建硬链接。

link oldfile newfile

ln oldfile newfile

由于硬链接是有着相同 inode 号仅文件名不同的文件，因此硬链接存在以下几点特性：

- 文件有相同的 inode 及 data block;
- 只能对已存在的文件进行创建;
- 不能交叉文件系统进行硬链接的创建;
- 不能对目录进行创建，只可对文件创建;
- 删除一个硬链接文件并不影响其他有相同 inode 号的文件。

软链接与硬链接不同，若文件用户数据块中存放的内容是另一文件的路径名的指向，则该文件就是软连接。软链接就是一个普通文件，只是数据块内容有点特殊。软链接有着自己的 inode 号以及用户数据块（见 图 2.）。因此软链接的创建与使用没有类似硬链接的诸多限制：

- 软链接有自己的文件属性及权限等；
- 可对不存在的文件或目录创建软链接；
- 软链接可交叉文件系统；
- 软链接可对文件或目录创建；
- 创建软链接时，链接计数 i_nlink 不会增加；
- 删除软链接并不影响被指向的文件，但若被指向的原文件被删除，则相关软连接被称为死链接（即 dangling link，若被指向路径文件被重新创建，死链接可恢复为正常的软链接）。

ln -s old.file soft.link

10 linux kill 进程时杀不掉的原因
（1）进程已经成为僵尸进程，当它的父进程将它回收或将它的父进程 kill 掉即可
（2）进程处于内核态，忽略所有信号处理，只能通过重启系统杀死
kill -9 中-9 是什么意思　9 是信号编号 9 对应 SIGKILL

11 linux 常用命令
查看进程占用资源 top　查看网络状态 netstat

查看日志的最后 100 行 tail –n 100 filename

ifconfig 和 ipconfig

查询哪些端口被占用 netstat –nultp，查询当前所有被占用的端口号

把一个文件中的字符 A 全部替换成字符 B
找出当前文件夹下最大的十个文件
找出指定文件后缀的包含指定字符串的文件名字
找出文件中包含指定字符串的前后 5 行

目录和文件的相关操作
pwd：显示当前目录
mkdir –p /home/bird/testing/test1 –p 可以不必依次建立目录
rmdir –r test 删除目录
cp ./aaa /tmp/bbb 把当前目录下的 aaa 文件复制到/tmp 下，并更名为 bbb
cp –r /etc/ /tmp 复制 etc 目录下的所有内容到/tmp 下
rm：删除文件或目录，默认删除文件，-r 删除目录
mv：移动或更名现有的文件或目录
mv /home/test /home/test2 /home/test 变为/home/test2
文本文件内容查看
cat：把一个文件连续输出在屏幕上
cat ./aaa 显示当前目录下 aaa 的内容
tac：从最后一行开始显示，tac 与 cat 是倒置的
head：只显示头几行

head -n 100 /etc/man.comfig 显示前 100 行
如果/etc/man.comfig 有 141 行，那么
head -n -100 /etc/man.comfig 显示前 41 行，后面 100 行不会打印出来了
tail：只显示尾几行
tail -n 100 /etc/man.config 显示末尾 100 行
tail -n +100 /etc/man.config 从 100 行开始显示到末尾
touch aaa 新建一个名为 aaa 的文件
cat aaa | grep root 把文件 aaa 中包含 root 的行的内容显示出来
也可见写成 grep root aaa
查看系统信息
df：列出文件系统的整体磁盘占用情况
ps：显示某个时间点程序的运行情况
top：持续侦测程序运作的状态

11 缓冲区溢出漏洞的基本原理
https://www.cnblogs.com/fanzhidongyzby/p/3250405.html

12 大端和小端
Big Endian：对于 12345678，在内存中是这样存放的
低地址……高地址
12 34 56 78
高位字节……低位字节
X86 是 little Endian

13 守护进程、僵尸进程（Zombie Process）和孤儿进程，怎么解决僵尸进程
孤儿进程：
如果父进程先退出,子进程还没退出那么子进程将被 托孤给 init 进程,这是子进程的父进程就是 init 进程(1 号进程)
僵尸进程：
如果我们了解过 linux 进程状态及转换关系,我们应该知道进程这么多状态中有一种状态是僵死状态,就是进程终止后进入僵死状态(zombie),等待告知父进程自己终止,后才能完全消失.但是如果一个进程已经终止了,但是其父进程还没有获取其状态,那么这个进程就称之为僵尸进程.僵尸进程还会消耗一定的系统资源,并且还保留一些概要信息供父进程查询子进程的状态可以提供父进程想要的信息.一旦父进程得到想要的信息,僵尸进程就会结束.
守护进程：
,守护进程就是在后台运行,不与任何终端关联的进程,通常情况下守护进程在系统启动时就在运行,它们以 root 用户或者其他特殊用户(apache 和 postfix)运行,并能处理一些系统级的任务.习惯上守护进程的名字通常以 d 结尾(sshd),但这些不是必须的.

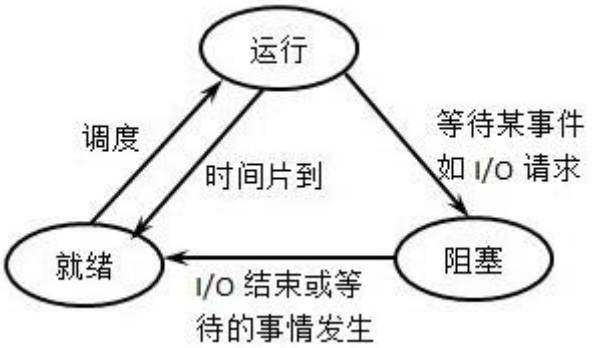
下面介绍一下创建守护进程的步骤

a) 调用 fork(),创建新进程,它会是将来的守护进程.
b) 在父进程中调用 exit,保证子进程不是进程组长
c) 调用 setsid()创建新的会话区
d) 将当前目录改成跟目录(如果把当前目录作为守护进程的目录,当前目录不能被卸载他作为守护进程的工作目录)

e) 将标准输入,标注输出,标准错误重定向到/dev/null

14 进程的常见状态？以及各种状态之间的转换条件？

- 就绪：进程已处于准备好运行的状态，即进程已分配到除 CPU 外的所有必要资源后，只要再获得 CPU，便可立即执行。
- 执行：进程已经获得 CPU，程序正在执行状态。
- 阻塞：正在执行的进程由于发生某事件（如 I/O 请求、申请缓冲区失败等）暂时无法继续执行的状态。



15 进程同步

进程同步的主要任务：是对多个相关进程在执行次序上进行协调，以使并发执行的诸进程之间能有效地共享资源和相互合作，从而使程序的执行具有可再现性。

同步机制遵循的原则：

（1）空闲让进；

（2）忙则等待（保证对临界区的互斥访问）；

（3）有限等待（有限代表有限的时间，避免死等）；

（4）让权等待，（当进程不能进入自己的临界区时，应该释放处理机，以免陷入忙等状态）。

16 进程的通信方式有哪些？

进程通信，是指进程之间的信息交换（信息量少则一个状态或数值，多者则是成千上万个字节）。因此，对于用信号量进行的进程间的互斥和同步，由于其所交换的信息量少而被归结为低级通信。

所谓高级进程通信指：用户可以利用操作系统所提供的一组通信命令传送大量数据的一种通信方式。操作系统隐藏了进程通信的实现细节。或者说，通信过程对用户是透明的。

1）共享存储器系统（存储器中划分的共享存储区）；实际操作中对应的是“剪贴板”（剪贴板实际上是系统维护管理的一块内存区域）的通信方式，比如举例如下：word 进程按下 ctrl+c，在 ppt 进程按下 ctrl+v，即完成了 word 进程和 ppt 进程之间的通信，复制时将数据放入到剪贴板，粘贴时从剪贴板中取出数据，然后显示在 ppt 窗口上。

（2）消息传递系统（进程间的数据交换以消息（message）为单位，当今最流行的微内核操作系统中，微内核与服务器之间的通信，无一例外地都采用了消息传递机制。应用举例：邮槽（MailSlot）是基于广播通信体系设计出来的，它采用无连接的不可靠的数据传输。邮槽是一种单向通信机制，创建邮槽的服务器进程读取数据，打开邮槽的客户机进程写入数据。

（3）管道通信系统（管道即：连接读写进程以实现他们之间通信的共享文件（pipe 文件，类似先进先出的队列，由一个进程写，另一进程读））。实际操作中，管道分为：匿名管道、命名管道。匿名管道是一个未命名的、单向管道，通过父进程和一个子进程之间传输数

据。匿名管道只能实现本地机器上两个进程之间的通信，而不能实现跨网络的通信。命名管道不仅可以在本机上实现两个进程间的通信，还可以跨网络实现两个进程间的通信。

- **管道**：**管道是单向的、先进先出的、无结构的、固定大小的字节流，它把一个进程的标准输出和另一个进程的标准输入连接在一起。**写进程在管道的尾端写入数据，读进程在管道的道端读出数据。数据读出后将从管道中移走，其它读进程都不能再读到这些数据。管道提供了简单的流控制机制。进程试图读空管道时，在有数据写入管道前，进程将一直阻塞。同样地，管道已经满时，进程再试图写管道，在其它进程从管道中移走数据之前，写进程将一直阻塞。

注 1：无名管道只能实现父子或者兄弟进程之间的通信，有名管道（FIFO）可以实现互不相关的两个进程之间的通信。

注 2：用 FIFO 让一个服务器和多个客户端进行交流时候，每个客户在向服务器发送信息前建立自己的读管道，或者让服务器在得到数据后再建立管道。使用客户的进程号（pid）作为管道名是一种常用的方法。客户可以先把自己的进程号告诉服务器，然后再到那个以自己进程号命名的管道中读取回复。

- **信号量**：**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，其它进程也访问该资源。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**：**是一个在系统内核中用来保存消　息的队列，它在系统内核中是以消息链表的形式出现的。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**。
- **共享内存**：共享内存允许两个或多个进程访问同一个逻辑内存。**这一段内存可以被两个或两个以上的进程映射至自身的地址空间中**，一个进程写入共享内存的信息，可以被其他使用这个共享内存的进程，通过一个简单的内存读取读出，从而实现了进程间的通信**。如果某个进程向共享内存写入数据，所做的改动将立即影响到可以访问同一段共享内存的任何其他进程。**共享内存是最快的 IPC 方式，它是针对其它进程间通信方式运行效率低而专门设计的。它往往与其它通信机制（如**信号量**）配合使用，来实现进程间的同步和通信。
- **套接字**：套接字也是一种进程间通信机制，与其它通信机制不同的是，它可用于不同机器间的进程通信。

17 上下文切换

对于单核单线程 CPU 而言，在某一时刻只能执行一条 CPU 指令。上下文切换(Context Switch)是一种**将 CPU 资源从一个进程分配给另一个进程的机制**。从用户角度看，计算机能够并行运行多个进程，这恰恰是操作系统通过快速上下文切换造成的结果。**在切换的过程中，操作系统需要先存储当前进程的状态(包括内存空间的指针，当前执行完的指令等等)，再读入下一个进程的状态，然后执行此进程。**

18 进程与线程的区别和联系？

- **进程**是具有一定独立功能的程序关于某个数据集合上的一次运行活动，**进程**是系统进行资源分配和调度的一个独立单位。
- **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。

进程和线程的关系

（1）一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。线程是操作系统可识别的最小执行和调度单位。

（2）资源分配给进程，同一进程的所有线程共享该进程的所有资源。 同一进程中的多个线程共享代码段(代码和常量)，数据段(全局变量和静态变量)，扩展段(堆存储)。但是每个线程拥有自己的栈段，栈段又叫运行时段，用来存放所有局部变量和临时变量。
（3）处理机分给线程，即真正在处理机上运行的是线程。
（4）线程在执行过程中，需要协作同步。不同进程的线程间要利用消息通信的办法实现同步。

进程与线程的区别?

（1）进程有自己的独立地址空间，线程没有
（2）进程是资源分配的最小单位，线程是 CPU 调度的最小单位
（3）进程和线程通信方式不同(线程之间的通信比较方便。同一进程下的线程共享数据（比如全局变量，静态变量），通过这些数据来通信不仅快捷而且方便，当然如何处理好这些访问的同步与互斥正是编写多线程程序的难点。而进程之间的通信只能通过进程通信的方式进行。）
（4）进程上下文切换开销大，线程开销小
（5）一个进程挂掉了不会影响其他进程，而线程挂掉了会影响其他线程
（6）对进程进程操作一般开销都比较大，对线程开销就小了

为什么进程上下文切换比线程上下文切换代价高?

进程切换分两步:
1.切换页目录以使用新的地址空间
2.切换内核栈和硬件上下文
对于 linux 来说，线程和进程的最大区别就在于地址空间，对于线程切换，第 1 步是不需要做的，第 2 是进程和线程切换都要做的。
切换的性能消耗:
1、线程上下文切换和进程上下问切换一个最主要的区别是线程的切换虚拟内存空间依然是相同的,但是进程切换是不同的。这两种上下文切换的处理都是通过操作系统内核来完成的。内核的这种切换过程伴随的最显著的性能损耗是将寄存器中的内容切换出。
2、另外一个隐藏的损耗是上下文的切换会扰乱处理器的缓存机制。简单的说，一旦去切换上下文，处理器中所有已经缓存的内存地址一瞬间都作废了。还有一个显著的区别是当你改变虚拟内存空间的时候，处理的页表缓冲（processor's Translation Lookaside Buffer (TLB)）或者相当的神马东西会被全部刷新，这将导致内存的访问在一段时间内相当的低效。但是在线程的切换中，不会出现这个问题。

19 进程调度

调度种类

- **高级调度**：(High-Level Scheduling)又称为作业调度，它决定把后备作业调入内存运行;
- **低级调度**：(Low-Level Scheduling)又称为进程调度，它决定把就绪队列的某进程获得 CPU;
- **中级调度**：(Intermediate-Level Scheduling)又称为在虚拟存储器中引入，在内、外存对换区进行进程对换。

非抢占式调度与抢占式调度

- **非抢占式**：分派程序一旦把处理机分配给某进程后便让它一直运行下去，直到进程完成或发生进程调度进程调度某事件而阻塞时，才把处理机分配给另一个进程。
- **抢占式**：操作系统将正在运行的进程强行暂停，由调度程序将 CPU 分配给其他就绪

进程的调度方式。

调度策略的设计

- **响应时间**: 从用户输入到产生反应的时间
- **周转时间**: 从任务开始到任务结束的时间

CPU 任务可以分为**交互式任务**和**批处理任务**，调度最终的目标是**合理的使用 CPU，使得交互式任务的响应时间尽可能短，用户不至于感到延迟，同时使得批处理任务的周转时间尽可能短，减少用户等待的时间。**

调度算法：

FIFO 或 First Come, First Served (FCFS)先来先服务

- 调度的顺序就是任务到达就绪队列的顺序。
- 公平、简单(FIFO 队列)、非抢占、不适合交互式。
- 未考虑任务特性，平均等待时间可以缩短。

Shortest Job First (SJF)

- 最短的作业(CPU 区间长度最小)最先调度。
- SJF 可以保证最小的平均等待时间。

Shortest Remaining Job First (SRJF)

- SJF 的可抢占版本，比 SJF 更有优势。
- SJF(SRJF): 如何知道下一 CPU 区间大小？根据历史进行预测: 指数平均法。

优先权调度

- 每个任务关联一个优先权，调度优先权最高的任务。
- 注意：优先权太低的任务一直就绪，得不到运行，出现“饥饿”现象。

Round-Robin(RR)轮转调度算法

- 设置一个时间片，按时间片来轮转调度（“轮叫”算法）
- 优点: 定时有响应，等待时间较短；缺点: 上下文切换次数较多；
- 时间片太大，响应时间太长；吞吐量变小，周转时间变长；当时间片过长时，退化为 FCFS。

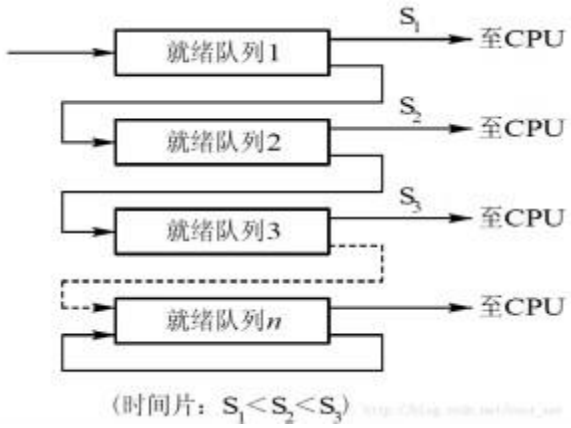
多级队列调度

- 按照一定的规则建立多个进程队列
- 不同的队列有固定的优先级（高优先级有抢占权）
- 不同的队列可以给不同的时间片和采用不同的调度方法
- 存在问题 1：没法区分 I/O bound 和 CPU bound；
- 存在问题 2：也存在一定程度的“饥饿”现象；

多级反馈队列

- **在多级队列的基础上，任务可以在队列之间移动，更细致的区分任务。**
- **可以根据“享用”CPU 时间多少来移动队列，阻止“饥饿”。**
- **最通用的调度算法，多数 OS 都使用该方法或其变形，如 UNIX、Windows 等。**

多级反馈队列调度算法描述:



- 进程在进入待调度的队列等待时，**首先进入优先级最高的 Q1 等待。**
- **首先调度优先级高的队列中的进程。若高优先级中队列中已没有调度的进程，则调度次优先级队列中的进程。**例如：Q1,Q2,Q3 三个队列，只有在 Q1 中没有进程等待时才去调度 Q2，同理，只有 Q1,Q2 都为空时才会去调度 Q3。
- **对于同一个队列中的各个进程，按照时间片轮转法调度。**比如 Q1 队列的时间片为 N，那么 Q1 中的作业在经历了 N 个时间片后若还没有完成，则进入 Q2 队列等待，若 Q2 的时间片用完后作业还不能完成，一直进入下一级队列，直至完成。
- **在低优先级的队列中的进程在运行时，又有新到达的作业，那么在运行完这个时间片后，CPU 马上分配给新到达的作业（抢占式）。**

一个简单的例子

假设系统中有 3 个反馈队列 Q1,Q2,Q3，时间片分别为 2，4，8。现在有 3 个作业 J1,J2,J3 分别在时间 0 ，1，3 时刻到达。而它们所需要的 CPU 时间分别是 3，2，1 个时间片。

- **时刻 0** J1 到达。 于是进入到队列 1 ，运行 1 个时间片 ，时间片还未到，此时 J2 到达。
- **时刻 1** J2 到达。 由于时间片仍然由 J1 掌控，于是等待。J1 在运行了 1 个时间片后，已经完成了在 Q1 中的 2 个时间片的限制，于是 J1 置于 Q2 等待被调度。现在处理机分配给 J2。
- **时刻 2** J1 进入 Q2 等待调度，J2 获得 CPU 开始运行。
- **时刻 3** J3 到达，由于 J2 的时间片未到，故 J3 在 Q1 等待调度，J1 也在 Q2 等待调度。
- **时刻 4** J2 处理完成，由于 J3，J1 都在等待调度，但是 J3 所在的队列比 J1 所在的队列的优先级要高，于是 J3 被调度，J1 继续在 Q2 等待。
- **时刻 5** J3 经过 1 个时间片，完成。
- **时刻 6** 由于 Q1 已经空闲，于是开始调度 Q2 中的作业，则 J1 得到处理器开始运行。 J1 再经过一个时间片，完成了任务。于是整个调度过程结束。

20 死锁的条件？以及如何处理死锁问题？

定义:如果一组进程中的每一个进程都在等待仅由该组进程中的其他进程才能引发的事件,那么该组进程就是死锁的。或者在两个或多个并发进程中，如果每个进程持有某种资源而又都等待别的进程释放它或它们现在保持着的资源，在未改变这种状态之前都不能向前推进，称这一组进程产生了死锁。通俗地讲，就是两个或多个进程被无限期地阻塞、相互等待的一种状态。

产生死锁的必要条件:

- **互斥条件(Mutual exclusion)**：资源不能被共享，只能由一个进程使用。
- **请求与保持条件(Hold and wait)**：已经得到资源的进程可以再次申请新的资源。
- **非抢占条件(No pre-emption)**：已经分配的资源不能从相应的进程中被强制地剥夺。
- **循环等待条件(Circular wait)**：系统中若干进程组成环路，该环路中每个进程都在等待相邻进程正占用的资源。

如何处理死锁问题：

- **忽略该问题**。例如鸵鸟算法，该算法可以应用在极少发生死锁的的情况下。为什么叫鸵鸟算法呢，因为传说中鸵鸟看到危险就把头埋在地底下，可能鸵鸟觉得看不到危险也就没危险了吧。跟掩耳盗铃有点像。
- **检测死锁并且恢复。**
- 仔细地对资源进行动态分配，使系统始终处于安全状态以**避免死锁**。
- **通过破除死锁四个必要条件之一，来防止死锁产生。**

21 临界资源

- 在操作系统中，进程是占有资源的最小单位（线程可以访问其所在进程内的所有资源，但线程本身并不占有资源或仅仅占有一点必须资源）。但**对于某些资源来说，其在同一时间只能被一个进程所占用。这些一次只能被一个进程所占用的资源就是所谓的临界资源**。典型的临界资源比如物理上的打印机，或是存在硬盘或内存中被多个进程所共享的一些变量和数据等(如果这类资源不被看成临界资源加以保护，那么很有可能造成丢数据的问题)。
- **对于临界资源的访问，必须是互斥进行。**也就是当临界资源被占用时，另一个申请临界资源的进程会被阻塞，直到其所申请的临界资源被释放。**而进程内访问临界资源的代码被成为临界区。**

22 一个程序从开始运行到结束的完整过程（四个过程）

1、预处理：条件编译，头文件包含，宏替换的处理，生成.i 文件。
2、编译：将预处理后的文件转换成汇编语言，生成.s 文件
3、汇编：汇编变为目标代码(机器代码)生成.o 的文件
4、链接：连接目标代码,生成可执行程序

23 内存池、进程池、线程池。(c++程序员必须掌握)

首先介绍一个概念“池化技术 ”。池化技术就是：提前保存大量的资源，以备不时之需以及重复使用。池化技术应用广泛，如内存池，线程池，连接池等等。内存池相关的内容，建议看看 Apache、Nginx 等开源 web 服务器的内存池实现。

由于在实际应用当做，分配内存、创建进程、线程都会设计到一些系统调用，系统调用需要导致程序从用户态切换到内核态，是非常耗时的操作。因此，当程序中需要频繁的进行内存申请释放，进程、线程创建销毁等操作时，通常会使用内存池、进程池、线程池技术来提升程序的性能。

线程池：线程池的原理很简单，类似于操作系统中的缓冲区的概念，它的流程如下：先启动若干数量的线程，并让这些线程都处于睡眠状态，当需要一个开辟一个线程去做具体的工作时，就会唤醒线程池中的某一个睡眠线程，让它去做具体工作，当工作完成后，线程又处于睡眠状态，而不是将线程销毁。

进程池与线程池同理。

内存池：内存池是指程序预先从操作系统申请一块足够大内存，此后，当程序中需要

申请内存的时候，不是直接向操作系统申请，而是直接从内存池中获取；同理，当程序释放内存的时候，并不真正将内存返回给操作系统，而是返回内存池。当程序退出(或者特定时间)时，内存池才将之前申请的内存真正释放。

24 虚拟内存？优缺点？

定义：具有请求调入功能和置换功能，能从逻辑上对内存容量加以扩充得一种存储器系统。其逻辑容量由内存之和和外存之和决定。

与传统存储器比较虚拟存储器有以下三个主要特征：

- 多次性，是指无需在作业运行时一次性地全部装入内存，而是允许被分成多次调入内存运行。
- 对换性，是指无需在作业运行时一直常驻内存，而是允许在作业的运行过程中，进行换进和换出。
- 虚拟性，是指从逻辑上扩充内存的容量，使用户所看到的内存容量，远大于实际的内存容量。

虚拟内存的实现有以下两种方式：

- 请求分页存储管理。
- 请求分段存储管理。

25 页面置换算法

操作系统将内存按照页面进行管理，在需要的时候才把进程相应的部分调入内存。当产生缺页中断时，需要选择一个页面写入。如果要换出的页面在内存中被修改过，变成了“脏”页面，那就需要先写会到磁盘。页面置换算法，就是要选出最合适的一个页面，使得置换的效率最高。页面置换算法有很多，简单介绍几个，重点介绍比较重要的 LRU 及其实现算法。

一、最优页面置换算法

最理想的状态下，我们给页面做个标记，挑选一个最远才会被再次用到的页面调出。当然，这样的算法不可能实现，因为不确定一个页面在何时会被用到。

二、先进先出页面置换算法（FIFO）及其改进

这种算法的思想和队列是一样的，该算法总是淘汰最先进入内存的页面，即选择在内存中驻留时间最久的页面予淘汰。实现：把一个进程已调入内存的页面按先后次序链接成一个队列，并且设置一个指针总是指向最老的页面。缺点：对于有些经常被访问的页面如含有全局变量、常用函数、例程等的页面，不能保证这些不被淘汰。

三、最近最少使用页面置换算法 LRU（Least Recently Used）

根据页面调入内存后的使用情况做出决策。LRU 置换算法是选择最近最久未使用的页面进行淘汰。

1.为每个在内存中的页面配置一个移位寄存器。（P165）定时信号将每隔一段时间将寄存器右移一位。最小数值的寄存器对应页面就是最久未使用页面。

2.利用一个特殊的栈保存当前使用的各个页面的页面号。每当进程访问某页面时，便将该页面的页面号从栈中移出，将它压入栈顶。因此，栈顶永远是最新被访问的页面号，栈底是最近最久未被访问的页面号。

26 中断与系统调用

所谓的中断就是在计算机执行程序的过程中，由于出现了某些特殊事情，使得 CPU 暂停对程序的执行，转而去执行处理这一事件的程序。等这些特殊事情处理完之后再回去执行之前的程序。中断一般分为三类：

- 由计算机硬件异常或故障引起的中断，称为**内部异常中断**;
- 由程序中执行了引起中断的指令而造成的中断，称为**软中断**（这也是和我们将要说明的系统调用相关的中断）;
- 由外部设备请求引起的中断，称为**外部中断**。简单来说，对中断的理解就是对一些特殊事情的处理。

与中断紧密相连的一个概念就是**中断处理程序**了。当中断发生的时候，系统需要去对中断进行处理，对这些中断的处理是由操作系统内核中的特定函数进行的，这些处理中断的特定的函数就是我们所说的中断处理程序了。

另一个与中断紧密相连的概念就是**中断的优先级**。中断的优先级说明的是当一个中断正在被处理的时候，处理器能接受的中断的级别。中断的优先级也表明了中断需要被处理的紧急程度。**每个中断都有一个对应的优先级，当处理器在处理某一中断的时候，只有比这个中断优先级高的中断可以被处理器接受并且被处理。**优先级比这个当前正在被处理的中断优先级要低的中断将会被忽略。

典型的中断优先级如下所示：

- **机器错误 > 时钟 > 磁盘 > 网络设备 > 终端 > 软件中断**

在讲系统调用之前，先说下**进程的执行在系统上的两个级别**：用户级和核心级，也称为**用户态和系统态(user mode and kernel mode)**。

用户空间就是用户进程所在的内存区域，相对的，**系统空间就是操作系统占据的内存区域**。用户进程和系统进程的所有数据都在内存中。**处于用户态的程序只能访问用户空间，而处于内核态的程序可以访问用户空间和内核空间。**

用户态切换到内核态的方式如下：

- **系统调用**：程序的执行一般是在用户态下执行的，但当程序需要使用操作系统提供的服务时，比如说打开某一设备、创建文件、读写文件（这些均属于系统调用）等，就需要向操作系统发出调用服务的请求，这就是系统调用。
- **异常**：当 CPU 在执行运行在用户态下的程序时，发生了某些事先不可知的异常，这时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，也就转到了内核态，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后，会向 CPU 发出相应的中断信号，这时 CPU 会暂停执行下一条即将要执行的指令转而去执行与中断信号对应的处理程序，如果先前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由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。

用户态和核心态(内核态）之间的区别是什么呢?

权限不一样。

- **用户态的进程能存取它们自己的指令和数据，但不能存取内核指令和数据（或其他进程的指令和数据）**。
- **核心态下的进程能够存取内核和用户地址某些机器指令是特权指令，在用户态下执行特权指令会引起错误。**在系统中内核并不是作为一个与用户进程平行的估计的进程的集合。

27 逻辑地址 Vs 物理地址 Vs 虚拟内存

- **所谓的逻辑地址，是指计算机用户(例如程序开发者)，看到的地址。**例如，当创建一个长度为 100 的整型数组时，操作系统返回一个逻辑上的连续空间：指针指向数组

第一个元素的内存地址。由于整型元素的大小为 4 个字节，故第二个元素的地址时起始地址加 4，以此类推。事实上，**逻辑地址并不一定是元素存储的真实地址，即数组元素的物理地址(在内存条中所处的位置)，并非是连续的，只是操作系统通过地址映射，将逻辑地址映射成连续的，这样更符合人们的直观思维**。

- 另一个重要概念是虚拟内存。操作系统读写内存的速度可以比读写磁盘的速度快几个量级。但是，内存价格也相对较高，不能大规模扩展。于是，**操作系统可以通过将部分不太常用的数据移出内存，"存放到价格相对较低的磁盘缓存，以实现内存扩展**。操作系统还可以通过算法预测哪部分存储到磁盘缓存的数据需要进行读写，提前把这部分数据读回内存。**虚拟内存空间相对磁盘而言要小很多，因此，即使搜索虚拟内存空间也比直接搜索磁盘要快。唯一慢于磁盘的可能是，内存、虚拟内存中都没有所需要的数据，最终还需要从硬盘中直接读取。**这就是为什么内存和虚拟内存中需要存储会被重复读写的数据，否则就失去了缓存的意义。现代计算机中有一个专门的**转译缓冲区(Translation Lookaside Buffer，TLB)**，用来实现虚拟地址到物理地址的快速转换。

与内存／虚拟内存相关的还有如下两个概念：

1) Resident Set

- 当一个进程在运行的时候，操作系统不会一次性加载进程的所有数据到内存，只会加载一部分正在用，以及预期要用的数据。其他数据可能存储在虚拟内存，交换区和硬盘文件系统上。**被加载到内存的部分就是 resident set。**

2) Thrashing

- 由于 resident set 包含预期要用的数据，理想情况下，进程运行过程中用到的数据都会逐步加载进 resident set。但事实往往并非如此：**每当需要的内存页面(page)不在 resident set 中时，操作系统必须从虚拟内存或硬盘中读数据，这个过程被称为内存页面错误(page faults)。当操作系统需要花费大量时间去处理页面错误的情况就是 thrashing。**

28 内部碎片与外部碎片

在内存管理中，**内部碎片**是已经被分配出去的的内存空间大于请求所需的内存空间。

外部碎片是指还没有分配出去，但是由于大小太小而无法分配给申请空间的新进程的内存空间空闲块。

固定分区存在内部碎片，可变式分区分配会存在外部碎片；

页式虚拟存储系统存在**内部碎片**；**段式虚拟存储**系统，存在**外部碎片**

为了有效的利用内存，使内存产生更少的碎片，要对内存分页，内存以页为单位来使用，最后一页往往装不满，于是形成了内部碎片。

为了共享要分段，在段的换入换出时形成外部碎片，比如 5K 的段换出后，有一个 4k 的段进来放到原来 5k 的地方，于是形成 1k 的外部碎片。

29 同步和互斥的区别

当有多个线程的时候，经常需要去同步这些线程以访问同一个数据或资源。例如，假设有一个程序，其中一个线程用于把文件读到内存，而另一个线程用于统计文件中的字符数。当然，在把整个文件调入内存之前，统计它的计数是没有意义的。但是，由于每个操作都有自己的线程，操作系统会把两个线程当作是互不相干的任务分别执行，这样就可能在没有把整个文件装入内存时统计字数。为解决此问题，你必须使两个线程同步工作。

所谓**同步**，是指散步在不同进程之间的若干程序片断，它们的运行必须严格按照规定的

某种先后次序来运行，这种先后次序依赖于要完成的特定的任务。如果**用对资源的访问来定义的话，同步是指在互斥的基础上（大多数情况），通过其它机制实现访问者对资源的有序访问。在大多数情况下，同步已经实现了互斥，特别是所有写入资源的情况必定是互斥的。少数情况是指可以允许多个访问者同时访问资源。**

所谓**互斥**，是指散布在不同进程之间的若干程序片断，当某个进程运行其中一个程序片段时，其它进程就不能运行它们之中的任一程序片段，只能等到该进程运行完这个程序片段后才可以运行。如果**用对资源的访问来定义的话，互斥某一资源同时只允许一个访问者对其进行访问，具有唯一性和排它性。但互斥无法限制访问者对资源的访问顺序，即访问是无序的。**

30 同步与异步

同步：

- 同步的定义：是指一个进程在执行某个请求的时候，若该请求需要一段时间才能返回信息，那么，这个进程将会一直等待下去，直到收到返回信息才继续执行下去。
- 特点：

1. 同步是阻塞模式；
2. 同步是按顺序执行，执行完一个再执行下一个，需要等待，协调运行；

异步：

- 是指进程不需要一直等下去，而是继续执行下面的操作，不管其他进程的状态。当有消息返回时系统会通知进程进行处理，这样可以提高执行的效率。
- 特点：

1. 异步是非阻塞模式，无需等待；
2. 异步是彼此独立，在等待某事件的过程中，继续做自己的事，不需要等待这一事件完成后再工作。线程是异步实现的一个方式。

同步与异步的优缺点：

- 同步可以避免出现死锁，读脏数据的发生。一般共享某一资源的时候，如果每个人都有修改权限，同时修改一个文件，有可能使一个读取另一个人已经删除了内容，就会出错，同步就不会出错。但，同步需要等待资源访问结束，浪费时间，效率低。
- 异步可以提高效率，但，安全性较低。

31 系统调用与库函数的区别

- **系统调用(System call)是程序向系统内核请求服务的方式。**可以包括硬件相关的服务(例如，访问硬盘等)，或者创建新进程，调度其他进程等。系统调用是程序和操作系统之间的重要接口。
- **库函数：把一些常用的函数编写完放到一个文件里，编写应用程序时调用，这是由第三方提供的，发生在用户地址空间**。
- 在**移植性方面**，不同操作系统的系统调用一般是不同的，移植性差；而在所有的 ANSI C 编译器版本中，C 库函数是相同的。
- 在**调用开销方面**，系统调用需要在用户空间和内核环境间切换，开销较大；而库函数调用属于“过程调用”，开销较小。

32 守护、僵尸、孤儿进程的概念

- **守护进程**：运行在后台的一种特殊进程，**独立于控制终端并周期性地执行某些任务**。
- **僵尸进程**：一个进程 fork 子进程，子进程退出，而父进程没有 wait/waitpid 子进程，

那么**子进程的进程描述符仍保存在系统中**，这样的进程称为僵尸进程。

- **孤儿进程**：一个**父进程退出，而它的一个或多个子进程还在运行**，这些子进程称为孤儿进程。（孤儿进程将由 init 进程收养并对它们完成状态收集工作）

33 Semaphore(信号量) Vs Mutex(互斥锁)

- 当用户创立多个线程 / 进程时，如果不同线程 / 进程同时读写相同的内容，则可能造成读写错误，或者数据不一致。此时，需要通过加锁的方式，控制临界区(critical section)的访问权限。对于 semaphore 而言，在初始化变量的时候可以控制允许多少个线程 / 进程同时访问一个临界区，其他的线程 / 进程会被堵塞，直到有人解锁。
- Mutex 相当于只允许一个线程 / 进程访问的 semaphore。此外，根据实际需要，人们还实现了一种读写锁(read-write lock)，它允许同时存在多个阅读者(reader)，但任何时候至多只有一个写者(writer)，且不能于读者共存。

34 IO 多路复用

IO 多路复用是指内核一旦发现进程指定的一个或者多个 IO 条件准备读取，它就通知该进程。IO 多路复用适用如下场合：

- 当客户处理多个描述字时（一般是交互式输入和网络套接口），必须使用 I/O 复用。
- 当一个客户同时处理多个套接口时，而这种情况是可能的，但很少出现。
- 如果一个 TCP 服务器既要处理监听套接口，又要处理已连接套接口，一般也要用到 I/O 复用。
- 如果一个服务器即要处理 TCP，又要处理 UDP，一般要使用 I/O 复用。
- 如果一个服务器要处理多个服务或多个协议，一般要使用 I/O 复用。
- 与多进程和多线程技术相比，I/O 多路复用技术的最大优势是系统开销小，系统不必创建进程/线程，也不必维护这些进程/线程，从而大大减小了系统的开销。

35 线程安全

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。或者说:一个类或者程序所提供的接口对于线程来说是原子操作或者多个线程之间的切换不会导致该接口的执行结果存在二义性,也就是说我们不用考虑同步的问题。

线程安全问题都是由全局变量及静态变量引起的。

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。

36 线程共享资源和独占资源问题

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。

用操作系统术语，线程切换的时候实际上切换的是一个可以称之为线程控制块的结构（TCB）,里面保存所有将来用于恢复线程环境必须的信息，包括所有必须保存的寄存器集，线程的状态等。

堆：　是大家共有的空间，分全局堆和局部堆。全局堆就是所有没有分配的空间，局部堆就是用户分配的空间。堆在操作系统对进程初始化的时候分配，运行过程中也可以向系统要额外的堆，但是记得用完了要还给操作系统，要不然就是内存泄漏。

栈：是个线程独有的，保存其运行状态和局部自动变量的。栈在线程开始的时候初始化，每个线程的栈互相独立，因此，栈是 thread safe 的。操作系统在切换线程的时候会自动的切换栈，就是切换 ＳＳ／ＥＳＰ寄存器。栈空间不需要在高级语言里面显式的分配和释放。

线程共享资源	线程独享资源
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
闹铃	
信号及信号服务程序	
记账信息	