操作系统

1.进程和线程

1.1 线程

1.1.1 概念

是进程中执行运算的最小单位，是进程中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有系统资源，只拥有一点在运行中必不可少的资源，但它可与同属一个进程的其它线程共享进程所拥有的全部资源。

1.1.2 好处

- 1. 易于调度。
- 2. 提高并发性。通过线程可方便有效地实现并发性。进程可创建多个线程来执行同一程序的不同部分。
- 3. 开销少。创建线程比创建进程要快，所需开销很少。。
- 4. 利于充分发挥多处理器的功能。通过创建多线程进程，每个线程在一个处理器上运行，从而实现应用程序的并发性，使每个处理器都得到充分运行。

1.1.3 线程状态

联系java中线程的几种状态
java thread的运行周期中, 有几种状态, 在 java.lang.Thread.State 中有详细定义和说明:

- **1)NEW** 状态是指线程刚创建, 尚未启动
- **2)RUNNABLE** 状态是线程正在正常运行中,当然可能会有某种耗时计算/IO等待的操作/CPU时间片切换等,这个状态下发生的等待一般是其他系统资源, 而不是锁, Sleep等
- **3)BLOCKED** 这个状态下, 是在多个线程有同步操作的场景,比如正在等待另一个线程的 synchronized 块的执行释放, 或者可重入的 synchronized块里别人调用wait() 方法,也就是这里是线程在等待进入临界区
- **4)WAITING** 这个状态下是指线程拥有了某个锁之后, 调用了他的wait方法, 等待其他线程/锁拥有者调用 notify / notifyAll 一遍该线程可以继续下一步操作, 这里要区分 BLOCKED 和 WATING 的区别, 一个是在临界点外面等待进入, 一个是在理解点里面wait等待别人notify, 线程调用了join方法 join了另外的线程的时候, 也会进入WAITING状态, 等待被他join的线程执行结束

- **5)TIMED_WAITING** 这个状态就是有限的(时间限制)的WAITING, 一般出现在调用wait(long), join(long)等情况下, 另外一个线程sleep后, 也会进入TIMED_WAITING状态
- **6) TERMINATED** 这个状态下表示该线程的run方法已经执行完毕了,基本上就等于死亡了(当时如果线程被持久持有, 可能不会被回收)

1.2 进程

1.2.1 概念

进程是操作系统的概念，每当我们执行一个程序时，对于操作系统来讲就创建了一个进程,在这个过程中，伴随着资源的分配和释放。可以认为进程是一个程序的一次执行过程。

1.2.2 进程与程序的区别

1. 进程是程序的一次运行活动，属于一种**动态**的概念。程序是一组有序的静态指令，是一种**静态**的概念。
2. 一个进程可以执行一个或多个程序。
3. 程序可以作为一种软件资源**长期**保持着,而进程则是一次执行过程,它是**暂时**的,是动态地产生和终止的。
4. 进程更能真实地描述并发,而程序不能。
5. 进程由程序和数据两部分组成，进程是竞争计算机系统有限资源的基本单位
6. 进程具有创建其他进程的功能；而程序没有。
7. 进程还具有**并发性和交往性**，这也与程序的**封闭性**不同

1.2.3 进程的几种状态

进程的基本状态及状态之间的关系

- **1)创建状态(New)**：进程正在创建过程中，还不能运行。操作系统在创建状态要进行的工作包括分配和建立进程控制块表项、建立资源表格(如打开文件表)并分配资源、加载程序并建立地址空间表等。
- **2)就绪状态(Ready)**：进程已获得除处理器外的所需资源，等待分配处理器资源；只要分配了处理器进程就可执行。就绪进程可以按多个优先级来划分队列。例如，当一个进程由于时间片用完而进入就绪状态时，排人低优先级队列；当进程由I／O操作完成而进入就绪状态时，排入高优先级队列。
- **3)执行状态**:进程已获得CPU，其程序正在执行。在单处理机系统中，只有一个进程处于执行状态；在多处理机系统中，则有多个进程处于执行状态。
- **4)阻塞状态**:正在执行的进程由于发生某事件而暂时无法继续执行时，便放弃处理机而处于暂停状态，亦即进程的执行受到阻塞，把这种暂停状态称为阻塞状态，有时也称为等待状态或封锁状态。致使进程阻塞的典型事件有：请求I/O，申请缓冲空间等。通常将这种处于阻塞状态的进程也排成一个队列。有的系统则根据阻塞原因的不同而把处于阻塞状态的进程排成多个队列。

- **5)退出状态(Exit)**: 进程已结束运行，回收除进程控制块之外的其他资源，并让其他进程从进程控制块中收集有关信息(如记帐和将退出代码传递给父进程)。

1.2.4 作业(进程)调度算法

1)先来先服务调度算法(FCFS)
每次调度都是从后备作业队列中选择一个或多个最先进入该队列的作业，将它们调入内存，为它们分配资源、创建进程，然后放入就绪队列。

2)短作业(进程)优先调度算法(SPF)
短作业优先(SJF)的调度算法是从后备队列中选择一个或若干个估计运行时间最短的作业，将它们调入内存运行。缺点:长作业的运行得不到保证

3)优先权调度算法(HPF)
当把该算法用于作业调度时，系统将从后备队列中选择若干个优先权最高的作业装入内存。当用于进程调度时，该算法是把处理机分配给就绪队列中优先权最高的进程，这时，又可进一步把该算法分成如下两种。
可以分为:

1. 非抢占式优先权算法
2. 抢占式优先权调度算法

4)高响应比优先调度算法(HRN)
每次选择高响应比最大的作业执行，响应比=(等待时间+要求服务时间)/要求服务时间。该算法同时考虑了短作业优先和先来先服务。

(1) 如果作业的等待时间相同，则要求服务的时间愈短，其优先权愈高，因而该算法有利于短作业。

(2) 当要求服务的时间相同时，作业的优先权决定于其等待时间，等待时间愈长，其优先权愈高，因而它实现的是先来先服务。

(3) 对于长作业，作业的优先级可以随等待时间的增加而提高，当其等待时间足够长时，其优先级便可升到很高，从而也可获得处理机。简言之，**该算法既照顾了短作业，又考虑了作业到达的先后次序，不会使长作业长期得不到服务**。因此，该算法实现了一种较好的折衷。当然，在利用该算法时，每要进行调度之前，都须先做响应比的计算，这会**增加系统开销**。

5)时间片轮转法（RR）
在早期的时间片轮转法中，系统将所有的就绪进程按先来先服务的原则排成一个队列，每次调度时，把CPU分配给队首进程，并令其执行一个时间片。时间片的大小从几ms到几百ms。当执行的时间片用完时，由一个计时器发出时钟中断请求，调度程序便据此信号来停止该进程的执行，并将它送往就绪队列的末尾；然后，再把处理机分配给就绪队列中新的队首进程，同时也让它执行一个时间片。这样就可以

保证就绪队列中的所有进程在一给定的时间内均能获得一时间片的处理机执行时间。换言之，系统能在给定的时间内响应所有用户的请求。

6)多级反馈队列调度算法
它是目前被公认的一种较好的进程调度算法。

(1) 应设置多个就绪队列，并为各个队列赋予不同的优先级。第一个队列的优先级最高，第二个队列次之，其余各队列的优先权逐个降低。该算法赋予各个队列中进程执行时间片的大小也各不相同，在优先权愈高的队列中，为每个进程所规定的执行时间片就愈小。例如，第二个队列的时间片要比第一个队列的时间片长一倍，……，第i+1个队列的时间片要比第i个队列的时间片长一倍。

(2) 当一个新进程进入内存后，首先将它放入第一队列的末尾，按FCFS原则排队等待调度。当轮到该进程执行时，如它能在该时间片内完成，便可准备撤离系统；如果它在一个时间片结束时尚未完成，调度程序便将该进程转入第二队列的末尾，再同样地按FCFS原则等待调度执行；如果它在第二队列中运行一个时间片后仍未完成，再依次将它放入第三队列，……，如此下去，当一个长作业(进程)从第一队列依次降到第n队列后，在第n 队列便采取按时间片轮转的方式运行。

(3) 仅当第一队列空闲时，调度程序才调度第二队列中的进程运行；仅当第1～(i-1)队列均空时，才会调度第i队列中的进程运行。如果处理机正在第i队列中为某进程服务时，又有新进程进入优先权较高的队列(第1～(i-1)中的任何一个队列)，则此时新进程将抢占正在运行进程的处理机，即由调度程序把正在运行的进程放回到第i队列的末尾，把处理机分配给新到的高优先权进程。

1.2.4 作业与进程的区别

一个进程是一个程序对某个数据集的执行过程，是分配资源的基本单位。作业是用户需要计算机完成的某项任务，是要求计算机所做工作的集合。一个作业的完成要经过作业提交、作业收容、作业执行和作业完成4个阶段。而进程是对已提交完毕的程序所执行过程的描述，是资源分配的基本单位。其主要区别如下。

1. 作业是用户向计算机提交任务的任务实体。在用户向计算机提交作业后，系统将它放入外存中的作业等待队列中等待执行。而进程则是完成用户任务的执行实体，是向系统申请分配资源的基本单位。任一进程，只要它被创建，总有相应的部分存在于内存中。
2. 一个作业可由多个进程组成，且必须至少由一个进程组成，反过来则不成立。
3. 作业的概念主要用在批处理系统中，像UNIX这样的分时系统中就没有作业的概念。而进程的概念则用在几乎所有的多道程序系统中。

1.3 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。
- (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。
- (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。线程是指进程内的一个执行单元,也是进程内的可调度实体.

1.4 进程与线程的区别

- **1) 基本单位** 线程作为调度和分配的基本单位，进程作为拥有资源的基本单位
- **2)并发性:** 不仅进程之间可以并发执行，同一个进程的多个线程之间也可并发执行
- **3)拥有资源:** 进程是拥有资源的一个独立单位，线程不拥有系统资源，只拥有一点在运行中必不可少的资源，但它可与同属一个进程的其它线程共享进程所拥有的全部资源.
- **4)系统开销:**由于创建或撤销进程时，系统都要为之分配或回收资源，如内存空间、I/O 设备等，因此操作系统所付出的开销远大于创建或撤销线程时的开销。类似地，在进行进程切换时，涉及当前执行进程 CPU 环境的保存及新调度进程 CPU 环境的设置。而线程切换时只需保存和设置少量寄存器内容，开销很小。此外，由于同一进程内的多个线程共享进程的地址空间，因此，这些线程之间的同步与通信非常容易实现，甚至无需操作系统的干预。
- **5)通信方面**: 进程间通讯有管道、信号量、信号消息队列、socket来维护，而线程间通过通道、共享内存、信号灯来进行通信。

2.1 IPC几种通信方式(进程间的通信方式)

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。

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，读进程在管道的首端读出数据。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。

管道有三种：

① 普通管道：有两个限制：一是只支持半双工通信方式，即只能单向传输；二是只能在父子进程之间使用；

② 流管道：去除第一个限制，支持双向传输；

③ 命名管道：去除第二个限制，可以在不相关进程之间进行通信。

2）命名管道 (named pipe)： 命名管道也是半双工的通信方式，它克服了管道没有名字的限制，并且它允许**无亲缘关系进程间**的通信。命令管道在文件系统中有对应的文件名，命名管道通过命令mkfifo或系统调用mkfifo来创建。 Microsoft SQL Server数据库默认安装后的本地连接使用的就是命名管道。MySQL在 Window环境下，如果需要两个进程在同一台服务器上通信可以使用命名管道，通过 --enable-named-pipe选项设置。

3）信号量(semophore)： 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。

4）消息队列(message queue)： 消息队列是由**消息的链表**结构实现，存放在内核中并由消息队列标识符标识。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。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。

5）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。除了用于进程通信外，进程还可以发送信号给进程本身。

6）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。MySQL内部通信也使用了共享内存的方式，可以通过配置文件添加 --shared-memory实现

7）套接字(socket)： 也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

3.死锁的必要条件，怎么处理死锁

3.1 死锁概念

是指两个或两个以上的进程在执行过程中，**由于竞争资源或者由于彼此通信而造成的一种阻塞的现象**，若无外力作用，它们都将无法推进下去。

3.2 活锁

活锁指的是任务或者执行者没有被阻塞，由于某些条件没有满足，导致一直重复尝试，失败，尝试，失败。

3.3 死锁条件

1. **互斥条件**:一个资源每次只能被一个进程使用
2. **不可剥夺条件**:进程已获得的资源，在未使用完之前，不能强行剥夺
3. **请求与保持条件**:一个进程因请求资源而阻塞时，对已获得的资源保持不放
4. **循环等待条件**:若干进程之间形成一种头尾相接的循环等待资源关系.

3.4 死锁预防

1. **破坏互斥条件**。允许某些进程(线程)同时访问某些资源，但有的资源不允许同时被访问如打印机等。
   例如假脱机打印机技术允许若干个进程同时输出，唯一真正请求物理打印机的进程是打印机守护进程。
2. **破坏不可抢占条件**:即允许进程强行从占有者那里夺取某些资源。这种预防方法实现起来困难，会降低系统性能。
3. **破坏占有且申请条件**。可以实行预先分配策略，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。如果当前进程所需的全部资源得不到满足，则不分配任何资源。只有当系统能够满足当前的全部资源得到满足时，才一次性将所有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重新申请资源的现象，因此不会发生死锁。但是有以下缺点：

- 在许多情况下，一个进程在执行之前不可能知道它所需的全部资源。这是由于进程在执行时是动态的，不可预测的。
- 资源利用率低。无论所分配资源何时用到，一个进程只有在占有所需的全部资源后才能执行。即使有些资源最后才被该进程用到一次，但该进程在生存期间一直占有它们，造成长期占有。
- 降低了进程的并发性。因为资源有限，又加上存在浪费，能分配到所需全部资源的进程个数必然少了。

4. **破坏循环等待条件**。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。所有进程对资源的请求必须严格按资源需要递增的顺序提出。进程占用小好资源，才能申请大号资源，就不会产生环路。这种策略与前面的策略相比，资源的利用率和系统吞吐量都有很大提高，但是也存在以下缺点：

- 限制了进程对资源的请求，同时系统给所有资源合理编号也是件困难事，并增加了系统开销。

3.4 死锁的避免

1. **银行家算法**：该算法需要检查申请者对资源的最大需求量，如果系统现存的各类资源可以满足申请者的请求，就满足申请者的请求。这样申请者就可很快完成其计算，然后释放它占用的资源，从而保证了系统中的所有进程都能完成，所以可避免死锁的发生。

3.5 死锁的解除

一旦检测出死锁，就应立即采取相应的措施，以解除死锁。
死锁解除的主要方法有：

- **资源剥夺法**。挂起某些死锁进程，并抢占它的资源，将这些资源分配给其他的死锁进程。但应防止被挂起的进程长时间得不到资源，而处于资源匮乏的状态。
- **撤销进程法**。强制撤销部分、甚至全部死锁进程并剥夺这些进程的资源。撤销的原则可以按进程优先级和撤销进程代价的高低进行。

- **进程回退法。**让一（多）个进程回退到足以回避死锁的地步，进程回退时自愿释放资源而不是被剥夺。要求系统保持进程的历史信息，设置还原点。

3.6 死锁检测与死锁恢复

死锁检测算法: 死锁检测的基本思想是，如果一个进程所请求的资源能够被满足，那么就让它执行，否则释放它拥有的所有资源，然后让其它能满足条件的进程执行。

4.内存管理方式：页存储、段存储、段页存储

4.1 分页存储管理

4.1.1 基本思想

将程序的逻辑地址空间划分为固定大小的**页(page)**，而物理内存划分为同样大小的**页框(page frame)或物理块**，每个物理块的大小一般取2的整数幂。程序加载时，可将任意一页放人内存中任意一个页框，这些页框不必连续，从而实现了离散分配。该方法需要CPU的硬件支持，来实现逻辑地址和物理地址之间的映射。在页式存储管理方式中地址结构由两部构成，前一部分是页号，后一部分为页内地址w（位移量）。

逻辑地址到物理地址变化原理：CPU中的内存管理单元(MMU)按逻辑页号通过查进程页表得到物理页框号，将物理页框号与页内地址相加形成物理地址(见图4-4)。

4.1.2 页式管理方式的优点

- 没有外碎片，每个内碎片不超过页大小，提高内存的利用率。
- 一个程序不必连续存放。
- 便于改变程序占用空间的大小(主要指随着程序运行，动态生成的数据增多，所要求的地址空间相应增长)。

4.1.3 缺点

1. 无论数据有多少，都只能按照页面大小分配，容易产生**内部碎片**（一个页可能填充不满，造成浪费。
2. 不能体现程序逻辑
3. 分页方式的缺点是页长与程序的逻辑大小不相关
4. 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。

4.2 分段存储

4.2.1 思想

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。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（写c程序时会用到），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。

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。
在段式虚拟存储系统中，虚拟地址由段号和段内地址组成，虚拟地址到实存地址的变换通过段表来实现。

在为某个段分配物理内存时，可以采用**首先适配法、下次适配法、最佳适配法等**方法。在回收某个段所占用的空间时，要注意将收回的空间与其相邻的空间合并。

在段式虚拟存储系统中，虚拟地址由段号和段内地址组成，虚拟地址到实存地址的变换通过段表来实现。

4.2.2 地址映射

在段式 管理系统中，整个进程的地址空间是**二维**的，即其逻辑地址由段号和段内地址两部分组成。为了完成进程逻辑地址到物理地址的映射，处理器会查找内存中的段表，由段号得到段的首地址，加上段内地址，得到实际的物理地址。这个过程也是由处理器的硬件直接完成的，操作系统只需在进程切换时，将进程段表的首地址装入处理器的特定寄存器当中。这个寄存器一般被称作段表地址寄存器。

4.2.3 分段存储方式的优缺点

分页对程序员而言是不可见的，而分段通常对程序员而言是可见的，因而分段为组织程序和数据提供了方便。与页式虚拟存储器相比，段式虚拟存储器有许多优点：

- 段的逻辑独立性使其易于**编译、管理、修改和保护**，也便于多道程序共享。
- 段长可以根据需要动态改变，允许自由调度，以便有效利用主存空间。
- 方便编程，分段共享，分段保护，动态链接，动态增长

因为段的长度不固定，段式虚拟存储器也有一些缺点：

- 主存空间分配比较麻烦。
- 容易在段间留下许多碎片（外部碎片），造成存储空间利用率降低。
- 由于段长不一定是2的整数次幂，因而不能简单地像分页方式那样用虚拟地址和实存地址的最低若干二进制位作为段内地址，并与段号进行直接拼接，必须用加法操作通过段起址与段内地址的求和运算得到物理地址。因此，**段式存储管理比页式存储管理方式需要更多的硬件支持。**

4.3 分页和分段的主要区别

- 页是信息的**物理单位**，分页是为实现离散分配方式，以消减内存的外零头，提高内存的利用率；段则是信息的**逻辑单位**，它含有一组其意义相对完整的信息，分段的目的是为了能更好地满足用户的需要。
- 页的**大小固定且由系统决定**，由系统把逻辑地址划分为页号和页内地址两部分，是由机器硬件实现的，因而在系统中只能有一种大小的页面；而**段的长度却不固定**，决定于用户所编写的程序，通常由编译程序在对源程序进行编译时，根据信息的性质来划分。
- **分页的作业地址空间是一维的，即单一的线性地址空间**，程序员只需利用一个记忆符，即可表示一个地址；而分段的作业地址空间则是二维的，程序员在标识一个地址是，即需给出段名，又需给出段内地址。
- 分页信息很难保护和共享、分段存储按逻辑存储所以容易实现对段的保存和共享。

4.4 段页存储

程序员按照分段系统的地址结构将地址分为段号与段内位移量，地址变换机构将段内位移量分解为页号和页内位移量。

为实现段页式存储管理，系统应为每个进程设置一个段表，包括每段的段号，该段的页表始址和页表长度。每个段有自己的页表，记录段中的每一页的页号和存放在主存中的物理块

它首先将程序按其逻辑结构划分为若干个大小不等的逻辑段，然后再将每个逻辑段划分为若干个大小相等的逻辑页。主存空间也划分为若干个同样大小的物理页。辅存和主存之间的信息调度以页为基本传送单位，每个程序段对应一个段表，每页对应一个页表。

段页式系统中，作业的**地址结构包含三部分的内容：段号，页号，页内位移量**

CPU访问时，段表指示每段对应的页表地址，每一段的页表确定页所在的主存空间的位置，最后与页表内地址拼接，确定CPU要访问单元的物理地址。

段页存储管理方式综合了段式管理和页式管理的优点，但需要经过两级查表才能完成地址转换，消耗时间多。

4.1.1 地址变换的过程

- 进行地址变换时，首先利用段号S，将它与段表长TL进行比较。若S<TL，表示未越界。
- 于是利用段表始址和段号来求出该段所对应的段表项在段表中的位置，从中得到该段的页表始址
- 利用逻辑地址中的段内页号P来获得对应页的页表项位置，从中读出该页所在的物理块号b
- 利用块号和页内地址来构成物理地址。

4.1.2 段页式存储管理的优缺点

优点:

- (1) 它提供了大量的虚拟存储空间。
- (2) 能有效地利用主存，为组织多道程序运行提供了方便。

缺点：

- (1) 增加了硬件成本、系统的复杂性和管理上的开消。
- (2) 存在着系统发生抖动的危险。
- (3) 存在着内碎片。
- (4) 还有各种表格要占用主存空间。

段页式存储管理技术对当前的大、中型计算机系统来说，算是最通用、最灵活的一种方案。

5. 什么是虚拟内存

物理内存:在应用中，自然是顾名思义，物理上，真实的插在板子上的内存是多大就是多大了。而在CPU中的概念，物理内存就是CPU的地址线可以直接进行寻址的内存空间大小。

虚拟内存:它使得应用程序认为它拥有连续的可用的内存(一个连续完整的地址空间),而实际上，它通常是被分隔成多个物理内存碎片，还有部分暂时存储在外部磁盘存储器上，在需要时进行数据交换。

6. 虚拟地址、逻辑地址、线性地址、物理地址的区别

虚拟地址：指的是由程序产生的由段选择符和段内偏移地址两个部分组成的地址。为什么叫它是虚拟的地址呢？因为这两部分组成的地址并没有直接访问物理内存，而是要通过分段地址的变换机构处理或映射后才会对应到相应的物理内存地址。

逻辑地址：指由程序产生的与段相关的偏移地址部分。不过有些资料是直接把逻辑地址当成虚拟地址，两者并没有明确的界限。

线性地址：指的是虚拟地址到物理地址变换之间的中间层，是处理器可寻指的内存空间（称为线性地址空间）中的地址。程序代码会产生逻辑地址，或者说是**段中的偏移地址，加上相应段的基地址就生成了一个线性地址**。如果启用了分页机制，那么线性地址可以再经过变换产生物理地址。若是没有采用分页机制，那么线性地址就是物理地址。

物理地址：指的是现在CPU外部地址总线上的寻址物理内存的地址信号，是地址变换的最终结果！

7. 操作系统如何进行分页调度

1. **先进先出算法(FIFO)**:这种算法的实质是，总是选择在主存中停留时间最长（即最老）的一页置换，即先进入内存的页，先退出内存。理由是：最早调入内存的页，其不再被使用的可能性比刚调入内

存的可能性大。

2. **最优置换算法（OPT）**:当调入新的一页而必须预先置换某个老页时，所选择的老页应是**将来不再被使用**，或者是在最远的将来才被访问。采用这种页面置换算法，保证有最少的缺页率。
   但是最优页面置换算法的实现是困难的，因为它需要人们预先就知道一个进程整个运行过程中页面走向的全部情况。
3. **LRU（最近最少使用）算法**
   如果一个数据在最近一段时间内使用次数很少，那么在将来一段时间内被使用的可能性也很小。
   LRU算法是经常采用的页面置换算法，并被认为是相当好的，但是存在如何实现它的问题。LRU算法需要实际硬件的支持。
4. **时钟算法()**:对于页替换算法，用于替换的候选帧集合看做一个循环缓冲区，并且有一个指针与之相关联。当某一页被替换时，该指针被设置成指向缓冲区中的下一帧。当需要替换一页时，操作系统扫描缓冲区，以查找使用位被置为0的一帧。每当遇到一个使用位为1的帧时，操作系统就将该位重新置为0；如果在这个过程开始时，缓冲区中所有帧的使用位均为0，则选择遇到的第一个帧替换；如果所有帧的使用位均为1,则指针在缓冲区中完整地循环一周，把所有使用位都置为0，并且停留在最初的位置上，替换该帧中的页。由于该算法循环地检查各页面的情况，故称为CLOCK算法，又称为最近未用(Not Recently Used, NRU)算法。

8. 磁盘调度算法

1.**先来先服务（FCFS）**，按访问请求到达的先后顺序服务。简单，公平，但是效率不高，相临两次请求可能会造成最内到最外柱面寻道，使磁头反复移动，增加了服务时间，对机器不利。

2.**最短寻道时间优先(SSTF)**，优先选择距当前磁头最近的访问请求进行服务，主要考虑寻道优先。改善了磁盘平均服务时间，但是造成某些访问请求长期等待得不到服务。

3.**扫描算法（SCAN）**，当设备无访问请求时，磁头不动；当有访问请求时，磁头按一个方向移动，在移动过程中对遇到的访问请求进行服务，然后判断该方向上是否还有访问请求，如果有则继续.

4.**循环扫描算法（CSCAN）**：循环扫描调度算法是在扫描算法的基础上改进的。磁臂改为单项移动，由外向里。当前位置开始沿磁臂的移动方向去选择离当前磁臂最近的哪个柱面的访问者。如果沿磁臂的方向无请求访问时，再回到最外，访问柱面号最小的作业请求。

Shell使用基础

什么时候使用shell

> #1、Shell 适用于开发小工具和包装脚本
>
> 2、充当浇水语言的角色，仅仅调用其他程序，或是极少的数据处理

> 3、如果需要使用hash、嵌套array，建议用其他编程语言实现
>
> 4、对性能要求较高的场景，建议用其他语言实现

变量

定义变量

```
# a 和 1 中间不要留有空格，例如：不能写成 a = 1
a=1
echo $a
```

引用变量

```
a=1

# 写法一：
echo $a

# 写法二（推荐）：
echo ${a}

# 用${}的好处是字符串可以拼接
echo ${a}b  # => 输出 1b
```

引号

```
# 尽量使用双引号，双引号可以输出变量和转义，但是单引号完全是字面量
a=1
echo "${a}" # => 输出 1
echo '${a}' # => 输出 ${a}
```

环境变量

```
# 环境变量没有要求必须大写，但是建议写成大写

export A=1
# 或
declare -x B=2

# 也可以把普通变量导出为环境变量
C=3
export C
```

定义常量

```
# 常量一般写在文件开头
readonly CONST1
# 或
declare -r CONST2
```

默认全局变量

```
# 查看当前所在的目录
$PWD

# 上一次所在的目录
$OLDPWD

# PATH 环境变量
$PATH
```

特殊变量

```
# 脚本名称
$0

# 脚本参数，$1是第一个参数...
$1
$2
$3

# 参数个数
$#

# 上一个命令的结果，0是正常结束，非0是有异常
$?
```

基本语法

条件判断

shell 的判断可以用 `[]`，也可以用 `[[]]`，但建议用 `[[]]`

```
if [[ $1 -eq "1" ]];then
    echo "ok"
elif [[ $1 -eq "2" ]];then
    echo "2333"
else
    echo "no"
```

判断可分为数字比较、字符串比较和文件判断

数字比较

判断相等：`=`、`==`、`-eq`（建议用 `==` 或者 `-eq`，语义比较明确）

判断不等：`!=`、`-ne`

判断大于：`>`（需要双括号 `(())`）、`-gt`（`[[]]` 即可）

判断小于：`<`（需要双括号）、`-lt`（`[[]]` 即可）

```
nb=4

(( $nb > 4 )) && echo "yes" || echo "no"
# 或
[[ $nb -gt 4 ]] && echo "yes" || echo "no"
```

判断大于等于：`-ge`

判断小于等于：`-le`

字符串判断

判断相等：`=`、`==`、`-eq`

判断不等：`!=`、`-ne`

判断大于：`>`（需要双括号 `(())`）、`-gt`（`[[]]` 即可）

判断小于：`<`（需要双括号）、`-lt`（`[[]]` 即可）

判断大于等于： `-ge`

判断小于等于： `-le`

判断字符串不为null（判断长度不为0）： `-n`

判断字符串为null（判断长度为0）： `-z`

```
str=""
# 未定义和长度为零的字符串都算空字符串
[[ -z $str ]] && echo "yes" || echo "not" # 输出 yes
[[ -n $str ]] && echo "yes" || echo "not" # 输出 not
```

文件判断

判断是否存在（不限制类型）： `-e`

判断**文件**是否存在： `-f`

判断**文件夹**是否存在： `-d`

逻辑运算符

取反： `!`

或： `||`

与： `&&`

```
a=1
[[ ! $a == 1 || 1=1 ]] && echo "yes" || echo "no" # 输出 => yes
[[ ! $a == 1 && 1=1 ]] && echo "yes" || echo "no" # 输出 => no
```

case语句

```
# | 相当于 or,
# ;; 相当于break, 必须有
# *) 相当于 default

case $1 in
 s|start)
        echo "start"
        ;;
 stop)
        echo "stop"
        ;;
 restart)
        echo "restart"
        ;;
 *)
        echo "Usage:[start|stop|reload]"
        exit 1
        ;;
esac
exit 0
```

循环

for 循环

```
# 写法一
# 输出当前文件夹下的所有文件
for f in `ls`;do
    echo $f
done

# 写法二
# 输出1-100的奇数
for i in {1..100..2};do
    echo $i
done

# 写法三
# 计算从1加到100
sum=0
for(( i=1;i<100;i++ ));do
                echo $(( sum+=i ))
done
```

while 循环

```
nb=0
while [[ $nb -lt 10 ]];do
    echo $(( nb+=1  ))
done
```

函数

注意：函数的返回值只能是数字

```
# 函数定义
function foo() {
    echo 'output'
}

# 函数调用
foo

# 函数参数
function sum(){
    echo $1
    echo $2
    res=$1+$2
    if [[ $res==3 ]];then
        return 0
    else:
        return -1
    fi
}

# 接收函数返回值用 $? ，函数的返回值只能是数字!
sum 1 2
echo $?  # 输出 => 0
```

shell错误处理

`set -e`：报错时退出

`set -u`：检测未定义变量

`set -x`：调试执行，输出shell脚本的原始内容，编写jenkins的脚本常用

```
set -x
echo "hello world"

# 输出:
# + echo "hello world"  # +号 后面是源文件内容 输出格式由 PS4 的环境变量决定
# hello world
```

Linux常用指令

自行百度=- =，自行背几个。如：top,ifconfig,cd,ls,mkdir,vim,find,tail,cp,mv,kill,clear等等……