操作系统

1. 操作系统特点

并发性、共享性、虚拟性、异步性（不确定性、随机性？）。

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

进程是程序的一次执行，该程序可以与其他程序并发执行；
进程有运行、阻塞、就绪三个基本状态（五态模型：新建、终止，运行，就绪，阻塞）；
进程调度算法：先来先服务调度算法（FCFS）、短作业优先调度算法（SJF）、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、和进程相比，它是一种非常"节俭"的多任务操作方式。在 linux 系统下，启动一个新的进程必须分配给它独立的地址空间，建立众多的数据表来维护它的代码段、堆栈段和数据段，这是一种"昂贵"的多任务工作方式。（资源）

2、运行于一个进程中的多个线程，它们之间使用相同的地址空间，而且线程间彼此切换所需时间也远远小于进程间切换所需要的时间。据统计，一个进程的开销大约是一个线程开销的 30 倍左右。（切换效率）

3、线程间方便的通信机制。对不同进程来说，它们具有独立的数据空间，要进行数据的传递只能通过进程间通信的方式进行，这种方式不仅费时，而且很不方便。线程则不然，由于同一进城下的线程之间贡献数据空间，所以一个线程的数据可以直接为其他线程所用，

这不仅快捷，而且方便。（通信）

除以上优点外，多线程程序作为一种多任务、并发的工作方式，还有如下优点：

1、使多 CPU 系统更加有效。操作系统会保证当线程数不大于 CPU 数目时，不同的线程运行于不同的 CPU 上。(CPU 设计保证)

2、改善程序结构。一个既长又复杂的进程可以考虑分为多个线程，成为几个独立或半独立的运行部分，这样的程序才会利于理解和修改。（代码易维护）

5. 进程调度算法

根据不同的环境来讨论调度算法

1. 批处理系统

先来先服务（first-come first serverd FCFS）有利于长作业，不利于短作业

短作业优先（short job first SJF）按照估计运行时间最短的顺序进行调度。长作业有可能被饿死，处于一直等待短作业执行完毕的状态。

最短剩余时间优先（short remaining time next SRTN）按照估计剩余时间最短的顺序进行调度

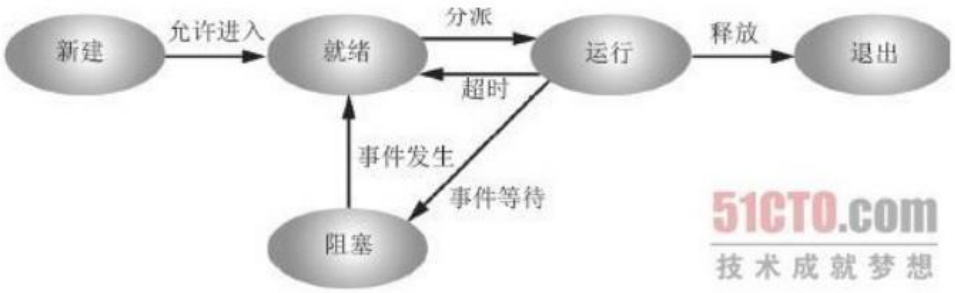
2. 交互式系统

时间片轮转：将所有就绪进程按照 FCFS 的原则排成一个队列，每次调度时，把 CPU 时间分配给队首进程，该进程可以执行一个时间片。当时间片用完时，由计时器发出时钟中断，调度程序便停止该进程的执行，并将它送往就绪队列的末尾，同时继续把 CPU 时间分配给队首的进程。

优先级调度：为每个进程分配一个优先级，按优先级进行调度。为了防止低优先级的进程永远等不到调度，可以随着时间的推移增加等待进程的优先级。

多级反馈队列：一个进程需要执行 100 个时间片，如果采用时间片轮转调度算法，那么需要交换 100 次。多级队列是为这种需要连续执行多个时间片的进程考虑，它设置了多个队列，每个队列时间片大小都不同，例如 1,2,4,8,...。进程在第一个队列没执行完，就会被移到下一个队列。这种方式下，之前的进程只需要交换 7 次。每个队列优先权也不同，最上面的优先权最高。因此只有上一个队列没有进程在排队，才能调度当前队列上的进程。

6. 进程状态转换图



1) 新建态：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

7. 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，fork 实现了写时拷贝. 而 vfork 直接让父子进程共用公用资源，避免多开辟空间拷贝，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；
4) 调用 fork()之后，父进程与子进程的执行顺序是我们无法确定的（即调度进程使用 CPU），意识到这一点极为重要，因为在一些设计不好的程序中会导致资源竞争，从而出现不可预知的问题。
5) 子进程会获得父进程所有文件描述符的副本
6) 一个进程能同时创建多少个线程这取决于地址空间的大小和内核参数，一台机器可以同时并发运行多少个线程也受限于 CPU 的数目
7) 在多线程执行的情况下调用 fork()函数，仅会将发起调用的线程复制到子进程中。（子进程中该线程的 ID 与父进程中发起 fork()调用的线程 ID 是一样的，因此，线程 ID 相同的情况有时我们需要做特殊的处理。）也就是说不能同时创建出于父进程一样多线程的子进程。其他线程均在子进程中立即停止并消失，并且不会为这些线程调用清理函数以及针对线程局部存储变量的析构函数。
8) fork()函数的调用会导致在子进程中除调用线程外的其它线程全都终止执行并消失，

因此在多线程的情况下会导致死锁和内存泄露的情况。在进行多线程编程的时候尽量避免 fork()的调用，同时在程序在进入 main 函数之前应避免创建线程，因为这会影响到全局对象的安全初始化。线程不应该被强行终止，因为这样它就没有机会调用清理函数来做相应的操作，同时也就没有机会来释放已被锁住的锁，如果另一线程对未被解锁的锁进行加锁，那么将会立即发生死锁，从而导致程序无法正常运行。

9) 推荐在多线程程序中调用 fork()的唯一情况是：其后立即调用 exec()函数执行另一个程序，彻底隔断子进程与父进程的关系。由新的进程覆盖掉原有的内存，使得子进程中的所有 pthreads 对象消失。

10) 从 fork 函数开始以后的代码父子共享，现在很多实现并不执行一个父进程数据段，堆和栈的完全复制. 而是采用写时拷贝技术。也就是如果你不修改我们一起用，你修改了之后对于修改的那部分内容我们分开各用个的.

8. 进程创建子进程,fork 详解

1) 函数原型

pid_t fork(void); //void 代表没有任何形式参数

2) 除了 0 号进程（系统创建的）之外，linux 系统中都是由其他进程创建的。创建新进程的进程，即调用 fork 函数的进程为父进程，新建的进程为子进程。

3) fork 函数不需要任何参数，对于返回值有三种情况：

① 对于父进程，fork 函数返回新建子进程的 pid；

② 对于子进程，fork 函数返回 0；

③ 如果出错， fork 函数返回 -1。

```
int pid=fork();
if(pid < 0){
//失败，一般是该用户的进程数达到限制或者内存被用光了
.....
}
else if(pid == 0){
//子进程执行的代码
.....
}
else{
```

```
//父进程执行的代码
.....
}
```

9. 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程序,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。
3) 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。
4) 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。

10. 进程和作业的区别?

1) 进程是程序的一次动态执行,属于动态概念;
2) 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;
3) 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;
4) 进程具有并发性,能与其他进程并发执行;
5) 进程是一个独立的运行单位;

11. 死锁是什么?必要条件?如何解决?

所谓死锁,是指多个进程因竞争资源而造成的一种僵局(互相等待)。很显然,如果没有外力的作用,那麼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。

〈1〉 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。

〈2〉 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。

〈3〉 占有且等待条件。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。

〈4〉 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 所占有的某一源，.....，而 Pn 等待 P1 所占有的的某一资源，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。

<1>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，这种办法并无实用价值。

<2>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。

<3>打破占有且等待条件。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。

<4>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

12. 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）。

13. 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。

设进程 cusneed 提出请求 REQUEST [i]，则银行家算法按如下规则进行判断。

(1)如果 REQUEST [cusneed] [i]<= NEED[cusneed][i]，则转（2）；否则，出错。

(2)如果 REQUEST [cusneed] [i]<= AVAILABLE[i]，则转（3）；否则，等待。

(3)系统试探分配资源，修改相关数据：

AVAILABLE[i]-=REQUEST[cusneed][i];

ALLOCATION[cusneed][i]+=REQUEST[cusneed][i];

NEED[cusneed][i]-=REQUEST[cusneed][i];

(4)系统执行安全性检查，如安全，则分配成立；否则试探险性分配作废，系统恢复原状，进程等待。

14. 进程间通信方式有几种，他们之间的区别是什么？

半双工和全双工：①当数据的发送和接收分流，分别由两根不同的传输线传送时，通信

双方都能在同一时刻进行发送和接收操作，这样的传送方式就是全双工制②若使用同一根传输线既作接收又作发送，虽然数据可以在两个方向上传送，但通信双方不能同时收发数据，这样的传送方式就是半双工制。

1) 管道

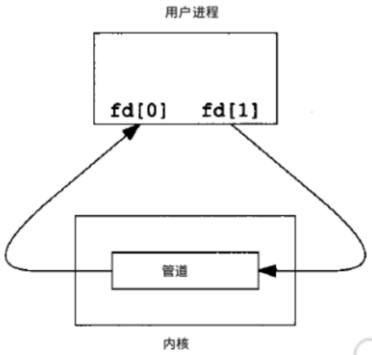
管道，通常指无名管道。

① 半双工的，具有固定的读端和写端；

② 只能用于具有亲属关系的进程之间的通信；

③ 可以看成是一种特殊的文件，对于它的读写也可以使用普通的 read、write 函数。但是它不是普通的文件，并不属于其他任何文件系统，只能用于内存中。

④ Int pipe(int fd[2]);当一个管道建立时，会创建两个文件文件描述符，fd[0]为读而打开，fd[1]为写而打开。要关闭管道只需将这两个文件描述符关闭即可。



2) FIFO（有名管道）

① FIFO 可以再无关的进程之间交换数据，与无名管道不同；

② FIFO 有路径名与之相关联，它以一种特殊设备文件形式存在于文件系统中；

③ Int mkfifo(const char* pathname,mode_t mode);

3) 消息队列

① 消息队列，是消息的链接表，存放在内核中。一个消息队列由一个标识符来标识；

② 消息队列是面向记录的，其中的消息具有特定的格式以及特定的优先级；

③ 消息队列独立于发送与接收进程。进程终止时，消息队列及其内容并不会被删除；

④ 消息队列可以实现消息的随机查询，消息不一定要以先进先出的次序读取，

4) 信号量

① 信号量是一个计数器，信号量用于实现进程间的互斥与同步，而不是用于存储进程间通信数据；

② 信号量用于进程间同步，若要在进程间传递数据需要结合共享内存；

③ 信号量基于操作系统的 PV 操作，程序对信号量的操作都是原子操作；

5) 共享内存

① 共享内存，指两个或多个进程共享一个给定的存储区；

② 共享内存是最快的一种进程通信方式（IPC），因为进程是直接对内存进行存取；

③ 因为多个进程可以同时操作，所以需要进行同步；

④ 信号量+共享内存通常结合在一起使用，信号量用来同步对共享内存的访问。

6) 五种通讯方式总结

① 管道：速度慢，容量有限，只有父子进程能通讯。

② FIFO：任何进程间都能通讯，但速度慢。

③ 消息队列：容量受到系统限制，且要注意第一次读的时候，要考虑上一次没有读完数据的问题

④ 信号量：不能传递复杂消息，只能用来同步

⑤ 共享内存区：能够很容易控制容量，速度快，但要保持同步，比如一个进程在写的时候，另一个进程要注意读写的问题，相当于线程中的线程安全，当然，共享内存区同样可以用作线程间通讯，不过没这个必要，线程间本来就已经共享了同一进程内的一块内存

15. 线程同步的方式？怎么用？

互斥锁、条件变量（wait()、notice()）、读写锁、信号量

1) 线程同步是指多线程通过特定的设置来控制线程之间的执行顺序，也可以说在线程之间通过同步建立起执行顺序的关系；

2) 主要四种方式，临界区、互斥对象、信号量、事件对象；其中临界区和互斥对象主要用于互斥控制，信号量和事件对象主要用于同步控制；

3) 临界区：通过对多线程的串行化来访问公共资源或一段代码，速度快、适合控制数据访问。在任意一个时刻只允许一个线程对共享资源进行访问，如果有多个线程试图访问公共资源，那么在有一个线程进入后，其他试图访问公共资源的线程将被挂起，并一直等到进入临界区的线程离开，临界区在被释放后，其他线程才可以抢占。

4) 互斥对象（mutex）：互斥对象和临界区很像，采用互斥对象机制，只有拥有互斥对象的线程才有访问公共资源的权限。因为互斥对象只有一个，所以能保证公共资源不会同时被多个线程同时访问。当前拥有互斥对象的线程处理完任务后必须将线程交出，以便其他线程访问该资源。

5) 信号量：它允许多个线程在同一时刻访问同一资源，但是需要限制在同一时刻访问此资源的最大线程数目。在用 CreateSemaphore()创建信号量时即要同时指出允许的最大资源计数和当前可用资源计数。一般是将当前可用资源计数设置为最大资源计数，每增加一个线程对共享资源的访问，当前可用资源计数就会减 1 ，只要当前可用资源计数是大于 0 的，就可以发出信号量信号。但是当前可用计数减小 到 0 时则说明当前占用资源的线程数已经达到了所允许的最大数目，不能在允许其他线程的进入，此时

的信号量信号将无法发出。线程在处理完共享资源后，应在离 开的同时通过 ReleaseSemaphore （）函数将当前可用资源计数加 1 。在任何时候当前可用资源计数决不可能大于最大资源计数。

6) 事件对象：通过通知操作的方式来保持线程的同步，还可以方便实现对多个线程的优先级比较的操作。

16. 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，既需要给出段名，又需要给出段的地址值。

4) 分页之所以是一维的，原因在于分页的大小是固定的，且页码之间是连续的，操作的时候只需给出一个地址，就能够根据所给地址的大小与页面大小计算出在页码和页内地址，粗略举例，比如页面大小是 4KB，给一个地址为 5000，可以算出所在页码是 2，页内地址是 5000-4000=1000，即在第二页的第 1000 个位置。而分段的因为每段的长度不一样，必须给出段码和段内地址

17. 孤儿进程和僵尸进程的区别？怎么避免这两类进程？守护进程？

1、 一般情况下，子进程是由父进程创建，而子进程和父进程的退出是无顺序的，两者之间都不知道谁先退出。即父进程永远无法预测子进程到底什么时候结束，于是就产生了孤儿进程和僵尸进程。

2、 孤儿进程，是指一个父进程结束以后，而他的一个或者多个子进程还在运行，那么那些子进程将会成为孤儿进程。孤儿进程将被 init 进程（进程号为 1）所收养，并由 init 进程对他们完成状态收集工作。

僵尸进程，是指一个进程使用 fork()创建子进程，如果子进程退出，而父进程并没有调用 wait 或 waitpid 获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中，这种进程称为僵尸进程。当一个进程完成他的工作终止后，他的父进程

需要调用 wait()或者 waitpid()系统调用取得子进程的终止状态。子进程退出时向父进程发送 SIGCHILD 信号，父进程处理 SIGCHILD 信号。在信号处理函数中调用 wait 进行处理僵尸进程。原理是将（父进程退出）子进程变成孤儿进程，从而其父进程变为 init 进程，通过 init 进程可以处理僵尸进程。

3、 区别：孤儿进程是父进程已退出，而子进程未提出；僵尸进程是父进程未退出，而子进程已退出。

4、 守护进程（ daemon）是指在后台运行，没有控制终端与之相连的进程。它独立于控制终端，通常周期性地执行某种任务 。守护进程脱离于终端是为了避免进程在执行过程中的信息在任何终端上显示并且进程也不会被任何终端所产生的终端信息所打断。

18. 守护进程是什么？怎么实现？

1. 守护进程（Daemon）是运行在后台的一种特殊进程。它独立于控制终端并且周期性地执行某种任务或等待处理某些发生的事件。守护进程是一种很有用的进程。
2. 守护进程特点
   1) 守护进程最重要的特性是后台运行。
   2) 守护进程必须与其运行前的环境隔离开来。这些环境包括未关闭的文件描述符，控制终端，会话和进程组，工作目录以及文件创建掩模等。这些环境通常是守护进程从执行它的父进程（特别是 shell）中继承下来的。
   3) 守护进程的启动方式有其特殊之处。它可以在 Linux 系统启动时从启动脚本/etc/rc.d 中启动，可以由作业规划进程 crond 启动，还可以由用户终端（shell）执行。
3. 实现
   1) 在父进程中执行 fork 并 exit 推出；
   2) 在子进程中调用 setsid 函数创建新的会话；
   3) 在子进程中调用 chdir 函数，让根目录 ”/” 成为子进程的工作目录；
   4) 在子进程中调用 umask 函数，设置进程的 umask 为 0；
   5) 在子进程中关闭任何不需要的文件描述符

19. 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) 每个独立的进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	

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 ID

20. 线程比进程具有哪些优势?

1) 进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；

21. 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；

22. 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 yeild 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；

23. 递归锁？

1) 线程同步能够保证多个线程安全访问竞争资源，最简单的同步机制是引入互斥锁。互斥锁为资源引入一个状态：锁定/非锁定。某个线程要更改共享数据时，先将其锁定，此时资源的状态为“锁定”，其他线程不能更改；直到该线程释放资源，将资源的状态变成“非锁定”，其他的线程才能再次锁定该资源。互斥锁保证了每次只有一个线程进行写入操作，从而保证了多线程情况下数据的正确性。

2) 读写锁从广义的逻辑上讲，也可以认为是一种共享版的互斥锁。如果对一个临界区大部分是读操作而只有少量的写操作，读写锁在一定程度上能够降低线程互斥产生的代价。

3) Mutex 可以分为递归锁(recursive mutex)和非递归锁(non-recursive mutex)。可递归锁也可称为可重入锁(reentrant mutex)，非递归锁又叫不可重入锁(non-reentrant mutex)。二者唯一的区别是，同一个线程可以多次获取同一个递归锁，不会产生死锁。而如果一个线程多次获取同一个非递归锁，则会产生死锁。

24. 用户态到内核态的转化原理？

1) 系统调用

这是用户态进程主动要求切换到内核态的一种方式，用户态进程通过系统调用申请使用操作系统提供的服务程序完成工作，比如前例中 fork()实际上就是执行了一个创建新进程的系统调用。而系统调用的机制其核心还是使用了操作系统为用户特别开放的一个中断来实现，例如 Linux 的 int 80h 中断。

2) 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。

3) 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。

25. 中断的实现与作用，中断的实现过程?

① 关中断，进入不可再次响应中断的状态，由硬件实现。
② 保存断点，为了在中断处理结束后能正确返回到中断点。由硬件实现。
③ 将中断服务程序入口地址送 PC，转向中断服务程序。可由硬件实现，也可由软件实现。
④ 保护现场、置屏蔽字、开中断，即保护 CPU 中某些寄存器的内容、设置中断处理次序、允许更高级的中断请求得到响应，实现中断嵌套。由软件实现。
⑤ 设备服务，实际上有效的中断处理工作是在此程序段中实现的。由软件程序实现
⑥ 退出中断。在退出时，又应进入不可中断状态，即关中断、恢复屏蔽字、恢复现场、开中断、中断返回。由软件实现。

26. 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。

27. CPU 中断

1) CPU 中断是什么

① 计算机处于执行期间；

② 系统内发生了非寻常或非预期的急需处理事件；

③ CPU 暂时中断当前正在执行的程序而转去执行相应的事件处理程序；

④ 处理完毕后返回原来被中断处继续执行；

2) CPU 中断的作用

① 可以使 CPU 和外设同时工作，使系统可以及时地响应外部事件；

② 可以允许多个外设同时工作，大大提高了 CPU 的利用率；

③ 可以使 CPU 及时处理各种软硬件故障。

28. 执行一个系统调用时，OS 发生的过程，越详细越好

1．执行用户程序(如:fork)

2． 根据 glibc 中的函数实现，取得系统调用号并执行 int $0x80 产生中断。

3． 进行地址空间的转换和堆栈的切换，执行 SAVE_ALL。(进行内核模式)

4． 进行中断处理，根据系统调用表调用内核函数。

5． 执行内核函数。

6． 执行 RESTORE_ALL 并返回用户模式

29. 函数调用和系统调用的区别?

1) 系统调用

① 操作系统提供给用户程序调用的一组特殊的接口。用户程序可以通过这组特殊接口来获得操作系统内核提供的服务；

② 系统调用可以用来控制硬件；设置系统状态或读取内核数据；进程管理，系统调用接口用来保证系统中进程能以多任务在虚拟环境下运行；

③ Linux 中实现系统调用利用了 0x86 体系结构中的软件中断；

2) 函数调用

① 函数调用运行在用户空间；

② 它主要是通过压栈操作来进行函数调用；

3) 区别

函数库调用	系统调用
在所有的ANSI C编译器版本中，C库函数是相同的	各个操作系统的系统调用是不同的
它调用函数库中的一段程序（或函数）	它调用系统内核的服务
与用户程序相联系	是操作系统的一个入口点
在用户地址空间执行	在内核地址空间执行
它的运行时间属于"用户时间"	它的运行时间属于"系统时间"
属于过程调用，调用开销较小	需要在用户空间和内核上下文环境间切换，开销较大
在C函数库libc中有大约300个函数	在UNIX中大约有90个系统调用
典型的C函数库调用：system fprintf malloc	典型的系统调用：chdir fork write brk；

30. 经典同步问题解法：生产者与消费者问题，哲学家进餐问题，读者写者问题。

31. 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；设置不得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。

32. 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。

4) 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 来为每一个线程创造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。

33. linux 文件系统

1. 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；
2. 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

34. 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。

2) 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）

3) 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。

4) 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 通知调用者。

1) 阻塞 I/O

应用程序调用一个 IO 函数，导致应用程序阻塞，等待数据准备好。 如果数据没有准备好，一直等待….数据准备好了，从内核拷贝到用户空间,IO 函数返回成功指示。

2) 非阻塞 I/O

我们把一个 SOCKET 接口设置为非阻塞就是告诉内核，当所请求的 I/O 操作无法完成时，不要将进程睡眠，而是返回一个错误。这样我们的 I/O 操作函数将不断的测试数据是否已经准备好，如果没有准备好，继续测试，直到数据准备好为止。在这个不断测试的过程中，会大量的占用 CPU 的时间。

3) I/O 复用

I/O 复用模型会用到 select、poll、epoll 函数，这几个函数也会使进程阻塞，但是和阻塞 I/O 所不同的的，这三个函数可以同时阻塞多个 I/O 操作。而且可以同时对多个读操作，多个写操作的 I/O 函数进行检测，直到有数据可读或可写时，才真正调用 I/O 操作函数。

4) 信号驱动 I/O

首先我们允许套接口进行信号驱动 I/O,并安装一个信号处理函数，进程继续运行并不阻塞。当数据准备好时，进程会收到一个 SIGIO 信号，可以在信号处理函数中调用 I/O 操作函数处理数据。

5) 异步 I/O

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的输入输出操作。

35. IO 复用的原理？零拷贝？三个函数？epoll 的 LT 和 ET 模式的理解。

1) IO 复用是 Linux 中的 IO 模型之一，IO 复用就是进程预先告诉内核需要监视的 IO 条件，使得内核一旦发现进程指定的一个或多个 IO 条件就绪，就通过进程进程处理，从而不会在单个 IO 上阻塞了。Linux 中，提供了 select、poll、epoll 三种接口函数来实现 IO 复用。
2) Select

select 的缺点：

① 单个进程能够监视的文件描述符的数量存在最大限制，通常是 1024。由于 select 采用轮询的方式扫描文件描述符，文件描述符数量越多，性能越差；

② 内核/用户空间内存拷贝问题，select 需要大量句柄数据结构，产生巨大开销；（何为句柄：任意进程，只要每打开一个对象，就会获得一个句柄，这个句柄用来标志对某个对象的一次打开，通过句柄，可以直接找到对应的内核对象。句柄本身是进程的句柄表中的一个结构体，用来描述一次打开操作）

③ Select 返回的是含有整个句柄的数组，应用程序需要遍历整个数组才能发现哪些句柄发生事件；

④ Select 的触发方式是水平触发，应用程序如果没有完成对一个已经就绪的文件描述符进行 IO 操作，那么每次 select 调用还会将这些文件描述符通知进程。

3) Poll

与 select 相比，poll 使用链表保存文件描述符，一你才没有了监视文件数量的限制，但其他三个缺点依然存在

4) Epoll

上面所说的 select 缺点在 epoll 上不复存在，epoll 使用一个文件描述符管理多个描述符，将用户关系的文件描述符的事件存放到内核的一个事件表中，这样在用户空间和内核空间的 copy 只需一次。Epoll 是事件触发的，不是轮询查询的。没有最大的并发连接限制，内存拷贝，利用 mmap（）文件映射内存加速与内核空间的消息传递。

区别总结：

1) 支持一个进程所能打开的最大连接数
   ① Select 最大 1024 个连接，最大连接数有 FD_SETSIZE 宏定义，其大小是 32 位整数表示，可以改变宏定义进行修改，可以重新编译内核，性能可能会影响；
   ② Poll 没有最大连接限制，原因是它是基于链表来存储的；
   ③ 连接数限数有上限，但是很大；
2) FD 剧增后带来的 IO 效率问题
   ① 因为每次进行线性遍历，所以随着 FD 的增加会造成遍历速度下降，效率降低；
   ② Poll 同上；
   ③ 因为 epool 内核中实现是根据每个 fd 上的 callback 函数来实现的，只有活跃的 socket 才会主动调用 callback，所以在活跃 socket 较少的情况下，使用 epoll 没有前面两者的现象下降的性能问题。
3) 消息传递方式
   ① Select 内核需要将消息传递到用户空间，都需要内核拷贝；
   ② Poll 同上；
   ③ Epoll 通过内核和用户空间共享来实现的。

epoll 的 LT 和 ET 模式的理解：

epoll 对文件描述符的操作有两种模式：LT(level trigger)和 ET(edge trigger)，LT 是默认模式。

区别：

LT 模式：当 epoll_wait 检测到描述符事件发生并将此事件通知应用程序，应用程序可以不立即处理该事件。下次调用 epoll_wait 时，会再次响应应用程序并通知此事件。

ET 模式：当 epoll_wait 检测到描述符事件发生并将此事件通知应用程序，应用程序必须立即处理该事件。如果不处理，下次调用 epoll_wait 时，不会再次响应应用程序并通知此事件。

36. 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**外碎片**.

3) 伙伴系统
4) 据可移动性组织页避免内存碎片

37. 递归的原理是啥？递归中遇到栈溢出怎么解决

1) 基本原理

第一：每一级的函数调用都有它自己的变量。

第二：每一次函数调用都会有一次返回，并且是某一级递归返回到调用它的那一级，而不是直接返回到 main()函数中的初始调用部分。

第三：递归函数中，位于递归调用前的语句和各级被调函数具有相同的执行顺序。例如在上面的程序中，打印语句#1 位于递归调用语句之前，它按照递归调用的顺序被执行了 4 次，即依次为第一级、第二级、第三级、第四级。

第四：递归函数中，位于递归调用后的语句的执行顺序和各个被调函数的顺序相反。例如上面程序中，打印语句#2 位于递归调用语句之后，其执行顺序依次是：第四级、第三级、第二级、第一级。（递归调用的这种特性在解决涉及到反向顺序的编程问题中很有用，下文会说到）

第五：虽然每一级递归都有自己的变量，但是函数代码不会复制。

第六：递归函数中必须包含终止递归的语句。通常递归函数会使用一个 if 条件语句或其他类似语句一边当函数参数达到某个特定值时结束递归调用，如上面程序的 if(n > 4)。

2) 用递归实现算法时，有两个因素是至关重要的：**递归式**和**递归边界；**
3) 函数调用时通过栈（Stack）来实现的，每当调用一个函数，栈就会加一层栈帧，函数返回就减一层栈帧。而栈资源有限，当递归深度达到一定程度后，就会出现意想不到的结果，比如堆栈溢出；
4) 利用循环函数或者栈加 while 循环来代替递归函数。

38. ++i 是否是原子操作

i++的操作分三步：

（1）栈中取出 i

（2）i 自增 1

（3）将 i 存到栈

所以 i++不是原子操作，上面的三个步骤中任何一个步骤同时操作，都可能导致 i 的值不正确自增

二.++i

在多核的机器上，cpu 在读取内存 i 时也会可能发生同时读取到同一值，这就导致两次自增，实际只增加了一次。

综上，我认为 i++和++i 都不是原子操作。

39. 缺页中断，页表寻址

1) 一个进程对应一个页表，分页存储机制，一个进程对应很多页，执行进程时并不是所有页装入内存中，部分装入内存，当需要的那页不存在内存中，将发生缺页中断，将需要的那页从外存中调入内存中；

2) 页表寻址，页分为页号（从 0 开始编号）与页内偏移地址，两个寄存器，页表基地址寄存器，页表长度寄存器，块表；页的大小相同，内存中的块与页大小相同，页大小相同，页在逻辑上连续在物理上不连续；

3) 调页算法：先进先出，最佳页面置换算法（OPT），最近最久未使用（NRU），最近最少使用置换算法（LRU），先进先出算法（FIFO）会导致 Baley 问题；抖动，页面在内存与外存中的频繁调页；

4) 程序局部性原理，时间局部性、空间局部性；

40. LRU 的实现

1) 用一个数组来存储数据，给每一个数据项标记一个访问时间戳，每次插入新数据项的时候，先把数组中存在的数据的时间戳自增，并将新数据时间戳置为 0 插入到数组中。每次访问数组中的数据项的时候，将被访问的数据项时间戳置为 0。当数组空间已经满时，将时间戳最大的数据项淘汰；

2) 利用一个链表来实现，每次新插入数据的时候将新数据插入到链表头部；每次缓存命中，则将数据移动到链表头部；那么当链表满时，就将链表尾部的数据丢弃；

3) 利用链表和 hashmap。当需要插入新的数据项 的时候，如果新数据命中，则把该节点放到链表头部，如果不存在，则将新数据放在链表头部。若缓存满了，则将链表尾部的节点删除。

41. 内存分区

1) 固态分区，分区大小固定，但并不一定相同；

2) 可变分区，分区大小动态变化，首先适配、最佳适配、最差适配、下一次适配；

42. 伙伴系统相关

1) 伙伴系统是一种经典的内存管理方法。Linux 伙伴系统的引入为内核提供了一种用于**分配一组连续的页**而建立的一种高效的分配策略，并有效的解决了外碎片问题。

2) Linux 中的内存管理的“页”大小为 4KB。把所有的空闲页分组为 11 个块链表，每个块链表分别包含大小为 1，2，4，8，16，32，64，128，256，512 和 1024 个连续页框的页块。最大可以申请 1024 个连续页，对应 4MB 大小的连续内存。每个页块的第一个页的物理地址是该块大小的整数倍。

3) 当向内核**请求分配(2^(i-1)，2^i]**数目的页块时，按照 2^i 页块请求处理。如果对应的块链表中没有空闲页块，则在更大的页块链表中找。当分配的页块中有多余的页时，伙伴系统根据多余的页框大小插入到对应的空闲页块链表中。

当**释放单页**的内存时，内核将其置于 CPU 高速缓存中，对很可能出现在 cache 的页，则放到“快表”的列表中。在此过程中，内核先判断 CPU 高速缓存中的页数是否超过一定“阈值”，如果是，则将一批内存页还给伙伴系统，然后将该页添加到 CPU 高速缓存中。

当**释放多页**的块时，内核首先计算出该内存块的伙伴的地址。**内核将满足以下条件的三个块称为伙伴**：(1)两个块具有相同的大小，记作 b。(2)它们的物理地址是**连续**的。(3)第一块的第一个页的**物理地址**是 2*(2^b)的倍数。如果找到了该内存块的伙伴，确保该伙伴的所有页都是空闲的，以便进行合并。内存继续检查合并后页块的“伙伴”并检查是否可以合并，依次类推。

4) 内核将已分配页分为以下三种不同的类型：

不可移动页：这些页在内存中有固定的位置，不能够移动。

可回收页：这些页不能移动，但可以删除。内核在回收页占据了太多的内存时或者内存短缺时进行页面回收。

可移动页：这些页可以任意移动，用户空间应用程序使用的页都属于该类别。它们是通过页表映射的。当它们移动到新的位置，页表项也会相应的更新。

43. I/O 控制方式

1) 直接 I/O（轮询）

程序查询方式也称为程序轮询方式，该方式采用用户程序直接控制主机与外部设备之间输入/输出操作。CPU 必须不停地循环测试 I/O 设备的状态端口，当发现设备处于准备好(Ready)状态时，CPU 就可以与 I/O 设备进行数据存取操作。这种方式下的 CPU 与 I/O 设备是串行工作的，输入/输出一般以字节或字为单位进行。这个方式频繁地测试 I/O 设备，I/O 设备的速度相对来说又很慢，极大地降低了 CPU 的处理效率，并且仅仅依靠测试设备状态位来进行数据传送，不能及时发现传输中的硬件错误。

2) 中断

当 I/O 设备结束(完成、特殊或异常)时，就会向 CPU 发出中断请求信号，CPU 收到信号就可以采取相应措施。当某个进程要启动某个设备时，CPU 就向相应的设备控制器发出一条设备 I/O 启动指令，然后 CPU 又返回做原来的工作。CPU 与 I/O 设备可以并行工作，与程序查询方式相比，大大提高了 CPU 的利用率。但是在中断方式下，同程序查询方式一样，也是以字节或字为单位进行。但是该方法大大降低了 CPU 的效率，因为当中断发生的非常频繁的时候，系统需要进行频繁的中断源识别、保护现场、中断处理、恢复现场。这种方法对于以“块”为存取单位的块设备，效率是低下的。

3) DMA

DMA 方式也称为直接主存存取方式，其思想是：允许主存储器和 I/O 设备之间通过“DMA 控制器(DMAC)”直接进行批量数据交换，除了在数据传输开始和结束时，整个过程无须 CPU 的干预。每传输一个“块”数据只需要占用一个主存周期。

4) 通道

通道(Channel)也称为外围设备处理器、输入输出处理机，是相对于 CPU 而言的。是一个处理器。也能执行指令和由指令的程序，只不过通道执行的指令是与外部设备相关的指令。是一种实现主存与 I/O 设备进行直接数据交换的控制方式，与 DMA 控制方式相比，通道所需要的 CPU 控制更少，一个通道可以控制多个设备，并且能够一次进行多个不连续的数据块的存取交换，从而大大提高了计算机系统效率。

44. Spooling 技术

1) 假脱机系统：在联机的情况下实现的同时外围操作的技术称为 SPOOLing 技术，或称为假脱机技术。

2) 组成

1. **输入井和输出井**：输入井和输出井的存储区域是在**磁盘**上开辟出来的。输入输出井中的数据一般以文件的形式组织管理，这些文件称之为井文件。一个文件仅存放某一个进程的输入或输出数据，所有进程的数据输入或输出文件链接成为一个输入输出队列。
2. **输入缓冲区和输出缓冲区**：输入缓冲区和输出缓冲区的存储区域是在**内存**中开辟出来的。主要用于缓和 CPU 和磁盘之间速度不匹配的矛盾。输入缓冲区用于暂存有输入设备传送的数据，之后再传送到输入井；输出缓冲区 同理。
3. **输入进程和输出进程**：输入进程也称为预输入进程，用于模拟脱机输入时的外围控制机，将用户要求的数据从输入设备传送到输入缓冲区，再存放到输入井。当 CPU 需要的时候，直接从输入井将数据读入内存。反之，输出的同理。
4. **井管理程序**：用于控制作业与磁盘井之间信息的交换。

3) 特点

① **提高了 I/O 的速度**：对数据执行的 I/O 操作，已从对低速 I/O 设备执行的 I/O 操作演变为对磁盘缓冲区中数据的存取，如同脱机输入输出一样，提高了 I/O 速度，缓和了 CPU 和低速的 I/Os 设备之间速度的不匹配的矛盾。

② **将独占设备改造成了共享设备**：因为在假脱机打印机系统中，实际上并没有为任何进程分配设备，而只是在磁盘缓冲区中为进程分配了一个空闲盘块和建立了一张 I/O 请求表。

③ **实现了虚拟设备功能**：宏观上，对于每一个进程而言，它们认为是自己独占了一个设备，即使实际上是多个进程在同时使用一台独占设备。也可以说，假脱机系统，实现了将独占设备变换为若干台对应的逻辑设备的功能。

45. 通道技术

1) 通道是独立于 CPU，专门用来负责数据输入/输出传输工作的处理机，对外部设备实现统一管理，代替 CPU 对输入/输出操作进行控制，从而使输入，输出操作可与 CPU 并行操作。

2) 引入通道的目的

为了使 CPU 从 I/O 事务中解脱出来，同时为了提高 CPU 与设备，设备与设备之间的并行工作能力

46. 共享内存的实现

1) 两个不同进程 A、B 共享内存的意思是，同一块物理内存被映射到进程 A、B 各自的进程地址空间。进程 A 可以即时看到进程 B 对共享内存中数据的更新，反之亦然。由于多个进程共享同一块内存区域，必然需要某种同步机制，互斥锁和信号量都可以。

2) 共享内存是通过把同一块内存分别映射到不同的进程空间中实现进程间通信。而共享内存本身不带任何互斥与同步机制，但当多个进程同时对同一内存进行读写操作时会破坏该内存的内容，所以，在实际中，同步与互斥机制需要用户来完成。

3) （1）共享内存就是允许两个不想关的进程访问同一个内存
（2）共享内存是两个正在运行的进程之间共享和传递数据的最有效的方式
（3）不同进程之间共享的内存通常安排为同一段物理内存
（4）共享内存不提供任何互斥和同步机制，一般用信号量对临界资源进行保护。
（5）接口简单

47. 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。

2) 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。

3) 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

4) 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。

5) 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。

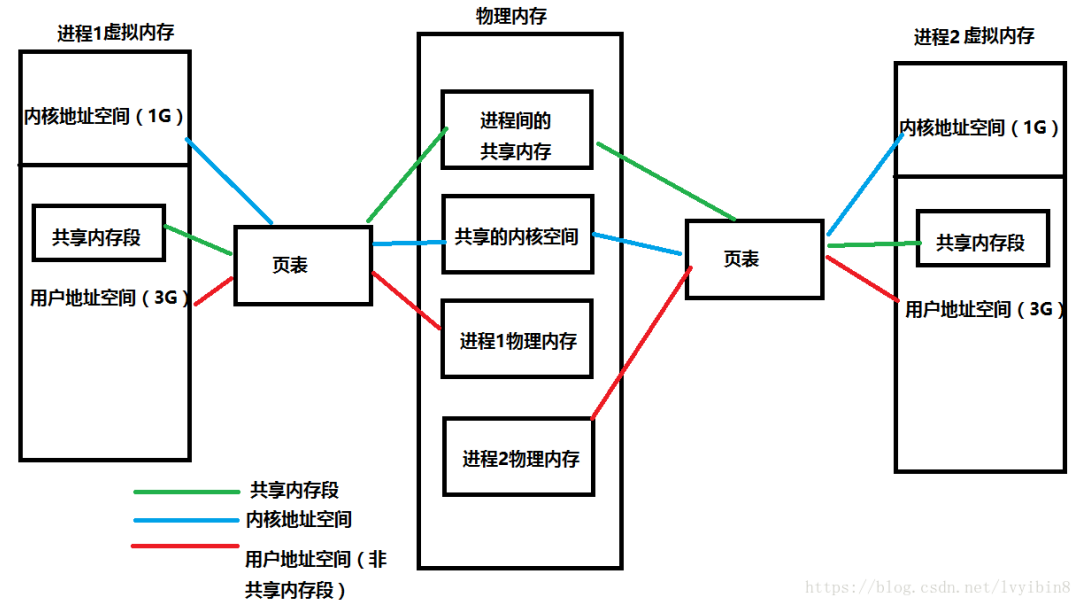
48. 虚拟内存和物理内存

1. **背景**:在没有虚拟内存概念时,程序寻址都是物理内存,程序能寻址的范围很有限。比如在 32bit 下，寻址的范围是 2^32 也就是 4G。如果没有虚拟内存，且每次启动一个进程都给 4G 的内存，出现很多问题：①内存有限，待分配的进程只能等待，效率低②直接访问物理内存，可以修改其他进程数据，修改内核空间数据③内存随机分配，程序运行的地址不正确。

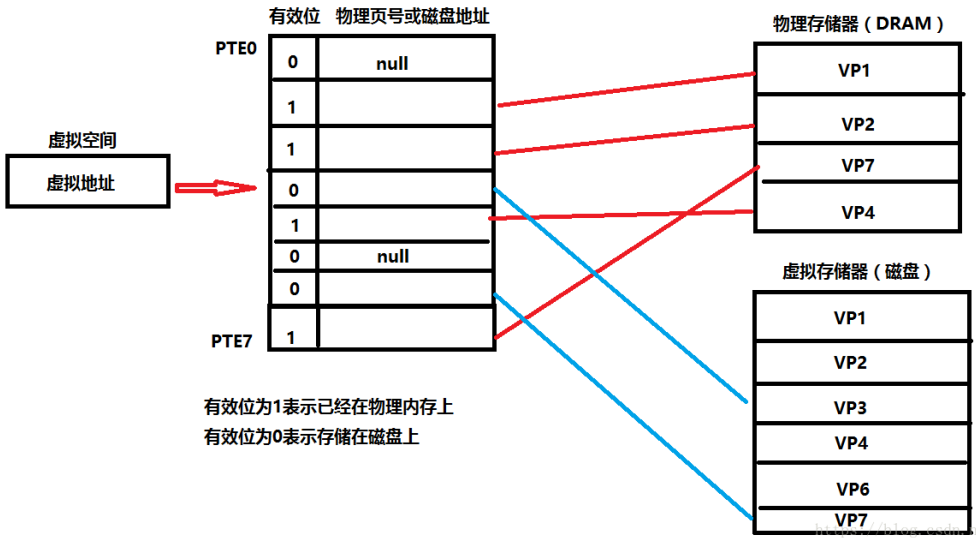
2. **虚拟内存**：每个进程都认为自己拥有 4G 的空间，这只是每个进程认为的，但是实际上，在虚拟内存对应的物理内存上，可能只对应的一点点的物理内存，实际用了多少内存，就会对应多少物理内存。进程得到的这 4G 虚拟内存是一个**连续的地址空间（这也只是进程认为），而实际上，它通常是被分隔成多个物理内存碎片，还有一部分存储在外部磁盘存储器上，在需要时进行数据交换。**

3. **进程访问地址的过程：**①每次我要访问地址空间上的某一个地址，都需要把地址翻译为实际物理内存地址②所有进程共享这整一块物理内存，每个进程只把自己目前需要的虚拟地址空间映射到物理内存上③进程需要知道哪些地址空间上的数据在物理内存上，哪些不在（可能这部分存储在磁盘上），还有在物理内存上的哪里，这就需要通过**页表**来记录④页表的每一个表项分两部分，第一部分记录此页是否在物理内存上，第二部分记录物理内存页的地址（如果在的话）⑤当进程访问某个虚拟地址的时候，就会先去看页表，如果发现对应的数据不在物理内存上，就会发生缺页异常⑥**缺页异常的处理过程**，操作系统立即阻塞该进程，并将硬盘里对应的页换入内存，然后使该进程就绪，如果内存已经满了，没有空地方了，那就找一个页覆盖，至于具体覆盖的哪个页，就需要看操作系统的页面置换算法是怎么设计的了。

4. **虚拟内存和物理内存的关系图**



5. **页表工作原理**



①我们的 cpu 想访问虚拟地址所在的虚拟页(VP3)，根据页表，找出页表中第三条的值.判断有效位。如果有效位为 1，DRMA 缓存命中，根据物理页号，找到物理页当中的内容，返回。②若有效位为 0，参数缺页异常，调用内核缺页异常处理程序。内核通过页面置换算法选择一个页面作为被覆盖的页面，将该页的内容刷新到磁盘空间当中。然后把 VP3 映射的磁盘文件缓存到该物理页上面。然后页表中第三条，有效位变成 1，第二部分存储上了可以对应物理内存页的地址的内容。③缺页异常处理完毕后，返回中断前的指令，重新执行，此时缓存命中，执行 1。④将找到的内容映射到告诉缓存当中，CPU 从告诉缓存中获取该值，结束。

6. 虚拟内存工作方式：当每个进程创建的时候，内核会为进程分配 4G 的虚拟内存，当进程还没有开始运行时，这只是一个内存布局。实际上并不立即就把虚拟内存对应位置的程序数据和代码（比如.text .data 段）拷贝到物理内存中，只是建立好虚

拟内存和磁盘文件之间的映射就好（叫做存储器映射）。这个时候数据和代码还是在磁盘上的。当运行到对应的程序时，进程去寻找页表，发现页表中地址没有存放在物理内存上，而是在磁盘上，于是发生缺页异常，于是将磁盘上的数据拷贝到物理内存中。另外在进程运行过程中，要通过 malloc 来动态分配内存时，也只是分配了虚拟内存，即为这块虚拟内存对应的页表项做相应设置，当进程真正访问到此数据时，才引发缺页异常。可以认为虚拟空间都被映射到了磁盘空间中（事实上也是按需要映射到磁盘空间上，通过 mmap，mmap 是用来建立虚拟空间和磁盘空间的映射关系的）

7. 虚拟内存优点：既然每个进程的内存空间都是一致而且固定的（32 位平台下都是 4G），所以链接器在链接可执行文件时，可以设定内存地址，而不用去管这些数据最终实际内存地址，这交给内核来完成映射关系

当不同的进程使用同一段代码时，比如库文件的代码，在物理内存中可以只存储一份这样的代码，不同进程只要将自己的虚拟内存映射过去就好了，这样可以节省物理内存

在程序需要分配连续空间的时候，只需要在虚拟内存分配连续空间，而不需要物理内存时连续的，实际上，往往物理内存都是断断续续的内存碎片。这样就可以有效地利用我们的物理内存

49. mmap

mmap 操作提供了一种机制，让用户程序直接访问设备内存，这种机制，相比较在用户空间和内核空间互相拷贝数据，效率更高。在要求高性能的应用中比较常用。mmap 映射内存必须是页面大小的整数倍，面向流的设备不能进行 mmap，mmap 的实现和硬件有关。

50. 内存外碎片和内碎片

1. 内部碎片：已经被分配出去（能明确指出属于哪个进程）却不能被利用的内存空间。内部碎片是处于(操作系统分配的用于装载某一进程的内存)区域内部或页面内部的存储块。占有这些区域或页面的进程并不使用这个存储块。而在进程占有这块存储块时，系统无法利用它。直到进程释放它，或进程结束时，系统才有可能利用这个存储块。

单道连续分配只有内部碎片。多道固定连续分配既有内部碎片，又有外部碎片。

2. 外部碎片：指的是还没有被分配出去（不属于任何进程），但由于太小了无法分配给申请内存空间的新进程的内存空闲区域。外部碎片是处于任何两个已分配区域或页面之间的空闲存储块。这些存储块的总和可以满足当前申请的长度要求，但是由于它们的地址不连续或其他原因，使得系统无法满足当前申请。**例如**：在内存上，分配三个**操作系统分配的用于装载进程的内存区域** A、B 和 C。假设，三个**内存区域**都是相连的。故而三个**内存区域**不会

产生外部碎片。现在假设 B 对应的进程执行完毕了操作系统随即收回了 B，这个时候 A 和 C 中间就有一块空闲**区域**了

51. 分段和分页内存管理、段页式存储管理

1. https://blog.csdn.net/beyond_2016/article/details/81358067

2. **两者优缺点**：在段式存储管理中，将程序的地址空间划分为若干段（segment），如代码段，数据段，堆栈段；这样每个进程有一个二维地址空间，相互独立，互不干扰。段式管理的优点是：没有内碎片（因为段大小可变，改变段大小来消除内碎片）。但段换入换出时，会产生外碎片（比如 4k 的段换 5k 的段，会产生 1k 的外碎片）

在页式存储管理中，将程序的逻辑地址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，程序加载时，可以将任意一页放入内存中任意一个页框，这些页框不必连续，从而实现了离散分离。页式存储管理的优点是：没有外碎片（因为页的大小固定），但会产生内碎片（一个页可能填充不满）。

3. **两者不同点**：(1) 分页仅仅是由于系统管理的需要而不是用户的需要。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。

(2) 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。

(3) 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时，既需给出段名，又需给出段内地址。

4. 在**段页式存储管理系统**中，作业的地址空间首先被分成若干个逻辑分段，每段都有自己的段号，然后再将每段分成若干个大小相等的页。对于主存空间也分成大小相等的页，主存的分配以页为单位。

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

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号。

5. **段页式存储管理系统的地址变换过程**：（1）程序执行时，从 PCB 中取出段表始址和段表长度，装入段表寄存器。（2）由地址变换机构将逻辑地址自动分成段号、页号和页内地址。（3）将段号与段表长度进行比较，若段号大于或等于段表长度，则表示本次访问的地址已超越进程的地址空间，产生越界中断。（4）将段表始址与段号和段表项长度

的乘积相加，便得到该段表项在段表中的位置。（5）取出段描述子得到该段的页表始址和页表长度。（6）将页号与页表长度进行比较，若页号大于或等于页表长度，则表示本次访问的地址已超越进程的地址空间，产生越界中断。（7）将页表始址与页号和页表项长度的乘积相加，便得到该页表项在页表中的位置。(8) 取出页描述子得到该页的物理块号。(9) 对该页的存取控制进行检查。（10）将物理块号送入物理地址寄存器中，再将有效地址寄存器中的页内地址直接送入物理地址寄存器的块内地址字段中，拼接得到实际的物理地址。

52. 页面置换算法

1. **最佳置换算法（OPT）（理想置换算法）**：从主存中移出永远不再需要的页面；如无这样的页面存在，则选择最长时间不需要访问的页面。于所选择的被淘汰页面将是以后永不使用的，或者是在最长时间内不再被访问的页面，这样可以保证获得最低的缺页率。

2. **先进先出置换算法（FIFO）**：是最简单的页面置换算法。这种算法的基本思想是：当需要淘汰一个页面时，总是选择驻留主存时间最长的页面进行淘汰，即先进入主存的页面先淘汰。其理由是：最早调入主存的页面不再被使用的可能性最大。

FIFO 算法还会产生当所分配的物理块数增大而页故障数不减反增的异常现象，这是由 Belady 于 1969 年发现，故称为 Belady 异常。只有 FIFO 算法可能出现 Belady 异常，而 LRU 和 OPT 算法永远不会出现 Belady 异常。

3. 最近最久未使用（LRU）算法：这种算法的基本思想是：利用局部性原理，根据一个作业在执行过程中过去的页面访问历史来推测未来的行为。它认为过去一段时间里不曾被访问过的页面，在最近的将来可能也不会再被访问。所以，这种算法的实质是：当需要淘汰一个页面时，总是选择在最近一段时间内最久不用的页面予以淘汰。

LRU 性能较好，但需要寄存器和栈的硬件支持。LRU 是**堆栈类**的算法。

53. PV 操作

PV 操作与信号量的处理相关，P 表示通过的意思，V 表示释放的意思。PV 操作由 P 操作原语和 V 操作原语组成（原语是不可中断的过程）

P（S）：①将信号量 S 的值减 1，即 S=S-1；②如果 S>=0，则该进程继续执行；否则该进程置为等待状态，排入等待队列。

V（S）：①将信号量 S 的值加 1，即 S=S+1；②如果 S>0，则该进程继续执行；否则释放队列中第一个等待信号量的进程

1. Inode 节点

1) Linux 操作系统引进了一个非常重要的概念 inode，中文名为索引结点，引进索引结点是为了在物理内存上找到文件块，所以 inode 中包含文件的相关基本信息，比如文件位置、文件创建者、创建日期、文件大小等待，输入 stat 指令可以查看某个文件的 inode 信息；
2) 硬盘格式化的时候，操作系统自动将硬盘分成两个区域，一个是数据区，一个是 inode 区，存放 inode 所包含的信息，查看每个硬盘分区的 inode 总数和已经使用的数量，可以用 df 命令；
3) 在 linux 系统中，系统内部并不是采用文件名查找文件，而是使用 inode 编号来识别文件。查找文件分为三个过程：系统找到这个文件名对应的 inode 号码，通过 inode 号码获得 inode 信息，根据 inode 信息找到文件数据所在的 block 读取数据；
4) 除了文件名之外的所有文件信息，都存储在 inode 之中。

2. Linux 软链接、硬链接，删除了软连接的源文件软连接可用?

1) 软链接可以看作是 Windows 中的快捷方式，可以让你快速链接到目标档案或目录。硬链接则透过文件系统的 inode 来产生新档名，而不是产生新档案。

2) 软链接（符号链接） ln –s 源文件 链接名
硬链接（实体链接） ln 源文件 链接名

3) 硬链接(hard link)：A 是 B 的硬链接（A 和 B 都是文件名），则 A 的目录项中的 inode 节点号与 B 的目录项中的 inode 节点号相同，即一个 inode 节点对应两个不同的文件名，两个文件名指向同一个文件，A 和 B 对文件系统来说是完全平等的。如果删除了其中一个，对另外一个没有影响。每增加一个文件名，inode 节点上的链接数增加一，每删除一个对应的文件名，inode 节点上的链接数减一，直到为 0，inode 节点和对应的数据块被回收。注：文件和文件名是不同的东西，rm A 删除的只是 A 这个文件名，而 A 对应的数据块（文件）只有在 inode 节点链接数减少为 0 的时候才会被系统回收。

4) 软链接(soft link)：A 是 B 的软链接（A 和 B 都是文件名），A 的目录项中的 inode 节点号与 B 的目录项中的 inode 节点号不相同，A 和 B 指向的是两个不同的 inode，继而指向两块不同的数据块。但是 A 的数据块中存放的只是 B 的路径名（可以根据这个找到 B 的目录项）。A 和 B 之间是“主从”关系，如果 B 被删除了，A 仍然存在（因为两个是不同的文件），但指向的是一个无效的链接。

5) 硬链接

不能对目录创建硬链接；不能对不同的文件系统创建硬链接；不能对不存在的文件创建硬链接；

6) 软连接

可以对目录创建软连接；可以跨文件系统；可以对不存在的文件创建软连接；

7) 因为链接文件包含有原文件的路径信息，所以当原文件从一个目录下移到其他目录中，再访问链接文件，系统就找不到了，而硬链接就没有这个缺陷，你想怎么移就怎么移；还有它要系统分配额外的空间用于建立新的索引节点和保存原文件的路径。

3. Linux 系统应用程序的内存空间是怎么分配的，用户空间多大，内核空间多大?

1) （32bit 系统）Linux 内核将这 4G 字节的空间分为两部分。将最高的 1G 字节（从虚拟地址 0xC0000000 到 0xFFFFFFFF），供内核使用，称为“内核空间”。而将较低的 3G 字节（从虚拟地址 0x00000000 到 0xBFFFFFFF），供各个进程使用，称为“用户空间“。因为每个进程可以通过系统调用进入内核。因此，Linux 内核由系统内的所有进程共享。于是，从具体进程的角度来看，每个进程可以拥有 4G 字节的虚拟空间。

4. Linux 的共享内存如何实现

1) 管道只能在具有亲缘关系的进程间进行通信；通过文件共享，在处理效率上又差一些，而且访问文件描述符不如访问内存地址方便；

2) mmap 内存共享映射，mmap 本来是存储映射功能，它可以将一个文件映射到内存中，在程序里就可以直接使用内存地址对文件内容进行访问；Linux 的 mmap 实现了一种可以在父子进程之间共享内存地址的方式；

3) XSI 共享内存，XSI 是 X/Open 组织对 UNIX 定义的一套接口标准（X/Open System Interface）。XSI 共享内存在 Linux 底层的实现实际上跟 mmap 没有什么本质不同，只是在使用方法上有所区别。

4) POSIX 共享内存，Linux 提供的 POSIX 共享内存，实际上就是在/dev/shm 下创建一个文件，并将其 mmap 之后映射其内存地址即可。

5. 文件处理 grep,awk,sed 这三个命令必知必会

1) grep

grep (global search regular expression(RE) and print out the line,全面搜索正则表达式并把行打印出来)是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。常用来在结果中搜索特定的内容。

2) awk

awk 是一个强大的文本分析工具，相对于 grep 的查找，sed 的编辑，awk 在其对数据分析并生成报告时，显得尤为强大。简单来说 awk 就是把文件(或其他方式的输入流,如重定向输入)逐行的读入（看作一个记录集），把每一行看作一条记录，以空格(或\t,或用户自己指定的分隔符)为默认分隔符将每行切片（类似字段），切开的部分再进行各种分析处理。

3) sed

sed 更侧重对搜索文本的处理，如修改、删除、替换等等。sed 主要用来自动编辑一个或多个文件；简化对文件的反复操作；编写转换程序等。

6. 查询进程占用 CPU 的命令

1) top

top 命令可以实时动态地查看系统的整体运行情况，是一个综合了多方信息监测系统性能和运行信息的实用工具。

2) ps

ps 命令就是最基本进程查看命令。使用该命令可以确定有哪些进程正在运行和运行的状态、进程是否结束、进程有没有僵尸、哪些进程占用了过多的资源等等.总之大部分信息都是可以通过执行该命令得到。ps 是显示瞬间进程的状态，并不动态连续；如果想对进程进行实时监控应该用 top 命令。

7. 一个程序从开始运行到结束的完整过程

1) 预处理，主要处理源代码中的预处理指令，引入头文件，去除注释，处理所有的条件编译指令，宏替换，添加行号。经过预处理指令后生成一个.i 文件；

2) 编译，编译过程所进行的是对预处理后的文件进行语法分析、词法分析、符号汇总，然后生成汇编代码。生成.s 文件；

3) 汇编，将汇编文件转换成二进制文件，二进制文件就可以让机器来读取。生成.o 文件；

4) 链接，由汇编程序生成的目标文件并不能立即就被执行，其中可能还有许多没有解决的问题。

8. 一般情况下在 Linux/windows 平台下栈空间的大小

windows 是编译器决定栈的大小，记录在可执行文件中，默认是 1M。linux 是操作系统来决定的，在系统环境变量中设置，ulimit -s 字节数 命令查看修改，但是 linux 默认栈大小为 10M;vs 编译器设置：属性—>设置→链接→输出→栈分配→重新设置；

9. Linux 重定向

1 重定向符号

>	输出重定向到一个文件或设备 覆盖原来的文件
>!	输出重定向到一个文件或设备 强制覆盖原来的文件
>>	输出重定向到一个文件或设备 追加原来的文件
<	输入重定向到一个程序

2 标准错误重定向符号

2>	将一个标准错误输出重定向到一个文件或设备 覆盖原来的文件 b-shell
2>>	将一个标准错误输出重定向到一个文件或设备 追加到原来的文件
2>&1	将一个标准错误输出重定向到标准输出 注释:1 可能就是代表 标准输出
>&	将一个标准错误输出重定向到一个文件或设备 覆盖原来的文件 c-shell
|&	将一个标准错误 管道 输送 到另一个命令作为输入

3 命令重导向示例

在 bash 命令执行的过程中，主要有三种输出入的状况，分别是：

1. 标准输入；代码为 0 ；或称为 stdin ；使用的方式为 <
2. 标准输出：代码为 1 ；或称为 stdout；使用的方式为 1>
3. 错误输出：代码为 2 ；或称为 stderr；使用的方式为 2>

10. Linux 常用命令

1) ls 命令，不仅可以查看 linux 文件包含的文件，而且可以查看文件权限；
2) cd 命令，切换当前目录到 dirName
3) pwd 命令，查看当前工作目录路径；
4) mkdir 命令，创建文件夹

5) rm 命令，删除一个目录中的一个或多个文件或目录

6) rmdir 命令，从一个目录中删除一个或多个子目录项，

7) mv 命令，移动文件或修改文件名

8) cp 命令，将源文件复制至目标文件，或将多个源文件复制至目标目录

9) cat 命令，显示文件内容；

10) touch 命令，创建一个文件

11) vim 命令，

12) which 命令查看可执行文件的位置，whereis 查看文件的位置，find 实际搜寻硬盘查询文件名称；

13) chmod 命令，用于改变 linux 系统文件或目录的访问权限，421，ewr

14) tar 命令，用来压缩和解压文件。tar 本身不具有压缩功能，只具有打包功能，有关压缩及解压是调用其它的功能来完成。

15) chown 命令，将指定文件的拥有者改为指定的用户或组，用户可以是用户名或者用户 ID;

16) ln 命令；

17) grep 命令，强大的文本搜索命令，grep 全局正则表达式搜素；

18) ps 命令，用来查看当前运行的进程状态，一次性查看，如果需要动态连续结果使用 top;

19) top 命令，显示当前系统正在执行的进程的相关信息，包括进程 ID、内存占用率、CPU 占用率等；

20) kill 命令，发送指定的信号到相应进程。不指定型号将发送 SIGTERM（15）终止指定进程。