名词解释

一.准备知识：

1. **操作系统**：
   系统就是抽象、协调和权衡。计算机操作系统建立起关于计算机上资源的分层抽象，并通过权衡利弊，统筹协调和管理资源来最大化计算机针对特定用途的效用。
   1. 地位：作为一切计算机软件的基石，统筹、协调和管理计算机系统中各类资源。
   2. 发展历程：
      批处理系统：按照一定顺序，逐一执行事先编组好的成批计算任务。
      分时系统：同时装入多个任务，每个任务分配一定的时间和空间运行。支持多用户交互。
      实时系统侧重严格时限内响应任务（如硬实时的导弹发射控制 ），解决特定场景需求 。
2. CPU流水线：
   取指-->译码-->执行-->回写

1.计算机基础：

1. 指令 ： 用编码表示CPU的一种操作，称为一条指令。
2. 指令系统：全部指令集称为指令系统。
3. CISC：复杂指令集计算机。（增强指令功能，设置功能复杂的指令，尽量使用硬件来实现功能）
4. RISC：精简指令集计算机（简化……，降低指令数量，……软件……）
5. 什么情况下需要隔离？程序之间如果不完全相互信任，就需要隔离。由操作系统负责。
   硬件实现隔离：内核模式与用户模式。

2.特权级：

1. 硬件特权级：CPU硬件实现，分为用户模式（用户态）和内核模式（内核态）。
2. 用户模式：执行应用程序的模式，不允许直接访问系统的敏感资源。
3. 内核模式：执行操作系统的模式，可直接执行敏感指令和访问敏感资源。
4. 特权指令 能对系统中敏感资源进行操作的指令，仅能在内核模式下执行。
5. 特权级的切换：中断机制或者专用指令机制。使用上述两种机制之一，之后陷入操作系统内核，执行被操作系统接管。

3.系统调用：

1. 系统调用：一种机制设计，允许应用程序通过受限的方法调用操作系统内核，向内核请求功能处理。

4.操作系统的结构：

库结构、宏内核结构、微内核结构、外核结构。

二.程序的结构：

1.简单的程序：

1.运行时视图：

1. 运行时视图：程序在执行时的主存储器布局。
2. 可执行文件：程序在外存上的储存方式，本质是保存了程序的逻辑布局的描述。
3. 程序的装入：操作系统读取外存上的可执行文件中对程序逻辑布局的描述，在内存中生成程序的物理布局的一个实例的过程。
4. 可执行文件头：描述程序运行时布局的元数据，一般附加在可执行文件头部。

2.程序的生成：

1. 编译器：把高级语言源程序翻译成机器语言程序。有时也先翻译成汇编语言源程序，然后调用汇编器。
2. 汇编器：把汇编语言源程序翻译成机器语言程序。
   汇编之后生成的.OBJ，这些文件还不能直接被执行，需要进一步链接，确定程序中所含地址的确切值后才可以直接被执行。
3. 链接器：将机器语言程序中间文件与系统运行库链接生成可执行的机器语言程序。可能重定位各个段的位置。

> *负责确定程序各部分所对应的具体地址，并以此填充所有符号引用、生成最终可执行二进制文件。*

链接器给予符号文件地址：直接回填法、间接地址法。
4. 调试器：系统提供给用户的能监督和控制用户程序的一种工具，可以装入、修改、显示或逐条执行一个程序。
5. 解释器：直接在机器上解释并执行高级语言源程序。解释器一般不会生成可独立运行的机器语言程序文件。

过程（子程序）结构：

程序指针可能在中途反复跳转到同一段过程执行，完毕后返回原处继续执行。
比循环结构更强大，因为过程的尾递归可以实现循环。过程还可以嵌套。
过程调用的全景：
调用开始-->保护寄存器-->读取参数-->初始化局部变量-->运算操作-->恢复寄存器-->调整栈框-->返回。

2.复杂的程序：

1.多工程共用：

1. 编程语言的库文件：在一门高级语言中，有大量基础功能是必备的，在所有程序中都很常用。于是他们将含有这些基础功能的文件预先编译好，形成大量的目标文件，程序员只要在生成程序的时候链接他们就可以了
2. 运行时环境：随着高级语言的发展，部分编程语言如Java等还要求垃圾回收、虚拟机管理、即时编译等功能，这些功能已经不由用户直接进行调用了，但它们仍然为语言运行提供基础的、关键的支持，必须随着程序的加载而被加载。因此，现代高级语言的运行时实际上是一种环境，不仅仅是库本身。运行时环境运行在用户态，它们也会使用系统调用。
3. 静态链接库：.OBJ文件的简单合集。它通常是将一堆*.OBJ文件的内容合并在一起成为一个文件，包括这些*.OBJ文件中包含的各个符号，以及各个符号的内容。

2.多道程序共享：

1. 动态链接库：在可执行程序被加载时，作为独立文件单独加载的库文件。它不包含在可执行文件之内；可执行文件将在需要它们的时候加载它们。
   判断一个库是不是动态链接库，就看调用它的可执行文件中是否包含它的内容。

   > *静态链接的问题：*
   > *1.重复存储 库会被在每一个可执行文件中保留一个副本，浪费磁盘。*
   > *2.更新困难 每次更新库之后，都需要把用到它的可执行文件重新链接。*
   > *3.重复加载 库会被在每一个虚拟地址空间中保留一个副本，进而导致在物理地址空间中也产生重复的副本，浪费内存，也浪费载入时间。*

2. 有了静态链接库，能否直接将它作为动态链接库用呢？
   将静态链接器做成静态链接库然后静态链接到应用程序中，应用程序在启动时先启动静态链接器对自己和库进行静态链接后再运行。

三.时间的协调:

1.处理器：

1. 指令流：一个应用程序内部可以由一个或多个逻辑上互相独立执行的指令序列组成。这种独立执行的指令序列叫做指令流。
2. 指令流的执行顺序：
   1. 顺序执行 一个指令流执行完成后，再去执行另一个指令流。
   2. 合作执行 将每个指令流打断成多份，每一份之内都顺序执行，但背靠背执行的两份不一定来自同一个指令流。在每份指令流的末尾，都通知操作系统主动放弃CPU，CPU将转去执行下一份指令流。

    3. 并发执行 将合作执行的条件放宽一点，允许一个指令流在任何时候被打断，并且新的指令流插入进来。又叫抢占式执行。
3. 指令流的五个状态：.....
4. 线程：操作系统提供给应用程序的一种对CPU时间的抽象机制。它是CPU时间分配的基本对象。应用程序通过将自己的指令流与线程对应起来，使指令流获得CPU时间分配。
   应用程序视角 在应用程序看来，自己的逻辑组织是一系列并发执行的指令流。
   操作系统视角 在操作系统看来，应用程序的运行组织是一系列被分配了CPU时间的线程。
5. 相比于指令流，线程的描述要包括什么数据？时间预算、优先级。
6. 线程控制块（TCB）：操作系统用以描述和管理线程的内核对象，一般至少包含线程的时间预算、优先级、运行状态及上下文，有时还会包含一些身份信息（如线程名、线程号）或统计信息（如总计CPU时间）等。*线程控制块位于内核空间。*
7. 指令流与线程的对应：.....
8. 协程和纤程：.....

处理器调度算法：

1. 固定优先级FP。（抢占/非抢占）
   先来先服务（FCFS）
   短作业优先（SJF）
   响应比高优先（HRRN）
   固定优先级时间片轮转法 （FPRR）
2. 公平的测度：吞吐量、等待时间、周转时间。
   正义的测速：响应时间、响应比。
3. 在线算法：算法必须在启动后逐个接受输入并即时给出输出。输入并非一次交给算法的，算法无法预测之后会遇到什么输入。
4. 饥饿现象：依照某种资源分配策略，某些请求无限增加时，另一些请求将永远得不到分配。

复杂系统的调度：

1. 多级队列 将线程按类型分成不同的队列，每个队列采用适合自身的调度算法，队列之间又有队列间的调度算法。
2. 多级反馈队列 在多级队列的基础上，实时动态检测每个线程的行为，并在原有队列不合适时为其更换到合适的队列。
3. 层次化调度：将一个系统分成多个子系统，每个子系统内部有自己的子调度器并采用适合自身的调度算法，一个父调度器则负责调度多个子系统。

多处理器并发：

并行执行 ： 多个指令流依附于多个位于不同物理处理器上的线程，做到了多个指令流的真正同时执行。

2.线程的实现：

内核线程 ：运行在内核空间的线程；它们运行时，CPU处于内核模式。
用户线程 ：运行在用户空间的线程；它们运行时，CPU处于用户模式。
"内核级"：指操作系统是否知道它们的存在
内核级线程：内核线程和用户线程都是内核级线程。
用户级线程：是指协程和纤程。

四.主存储器分配：

1.内存的分配：（分配动态内存——堆）

程序内分配：

1. 固定分区法、多固定块法、动态分区法（将整个堆切成与某个粒度相同的小块，粒度由所有分配的最大公约数决定）、首次适配法、最好适配法、最坏适配法。
   碎片：出于某些原因，无法有效利用而被浪费掉的资源。
   内部碎片：实际上已经指派给某个分配，但是逻辑上无法被这个分配利用的资源。
   外部碎片：尚未被实际分配出去，但因为某些逻辑上的原因无法分配的资源。
   问题：遍历链表、策略单一、粒度过细。

程序间分配：

内部碎片：指的是操作系统已经指派给程序，但逻辑上无法被这个程序利用的内存。
外部碎片：出现操作系统仍有内存但因为某些原因无法分配给应用程序的情况。

进程与内存隔离：

1. 进程：操作系统提供给应用程序的一种对地址空间的抽象机制。它是内存分配的基本对象。是计算机资源分配的独立单位。（内存加权限）
   应用程序视角 在应用程序看来，自己的逻辑组织是一系列段。
   操作系统视角 在操作系统看来，应用程序的空间组织是一个或一系列进程。
2. 保护域：是一组权能的集合，又称为权能空间。这些权能一方面赋予进程合法资源操作的权限，另一方面限制进程非法操作资源的企图，最终实现保护域之间的隔离。
   权能：一种代表对某种资源做某种操作的许可的不可伪造的令牌。
3. PCB：操作系统用以描述和管理进程的内核对象，一般至少包含进程的地址空间描述符表及一些其他权限表，有时还会包含一些身份信息（如进程名、进程号）、统计信息（如当前正在运行的线程数、总计内存大小）、线程信息（当前的线程列表，内含指向各个TCB的指针）等。进程控制块总是位于内核空间。
4. 可执行文件与进程：
   可执行文件：应用程序在外存上的存储方式。它描述了应该为应用程序建立一个（或一些）什么样的进程、进程中要有什么样的线程，以及线程和具体的指令流如何对应。它是死的、

干瘪的、静态的应用程序，没有执行环境和上下文，也没有执行活动。进程应用.程序在内存中的活动组织。它是活的、丰满的、动态的应用程序，具备一个由地址空间和其它权限提供的执行环境，并充满了线程（或说依附于线程上的指令流）的执行活动和上下文。
关系：可执行文件对进程为一对多关系。一个可执行文件每启动一次就可以创建一个（这是通常的实现）或一组进程；如果它启动多次，就可以创建一系列或一系列组进程。
同一个可执行文件，在启动为不同的进程时，可以处理不同的工作、使用不同的权限，或者以不同用户的名义启动。生成的多个进程之间是不同的，因为他们内部的执行环境、执行活动和内部线程的上下文均有差别。

5. 进程与线程：
   线程：CPU执行时间的分配对象，指令流通过依附于它获得执行时间。但它又需要依附在进程上获得执行空间。
   进程：仅仅一个执行空间，本身不具备执行能力。
   进程对线程可以为一对一、一对多、多对一、多对多关系。.....共享资源。

内存隔离机制+替换算法：

分段：

段表的优缺点:(在内存中)
快表（TLB）;

分页：

缺页异常的处理：
替换算法：LFD、FIFO、LRU、LFU
Belady异常:在某些资源分配策略下，增加资源总量反而导致性能下降和效率降低的现象。(FIFO)
抖动：当被分配的页数小于当前工作集的时候，缺页率会大幅增长。此时，程序的访存性能向外存的性能急剧跌落。

五.外存储器的分配：

1. 文件：一个具备一些属性的数据记录。
2. 文件系统 将文件中的文件块按照一定的方法最终映射到物理设备上的物理块，并在物理存储器特性的限制下提供尽可能高的信息管理效率。提供这种功能的软件栈称为文件系统。
3. 路径：文件在文件系统中的位置用路径表示。
4. 目录：文件路径中由分隔符隔开的子字符串，又称文件夹。
5. 相对路径 从某个目录D起始，经由逐步查找能找到某文件F的路径，称为相对于D的路径。
6. 绝对路径 某文件F相对于系统根目录的路径。
7. 工作目录 某应用程序进行文件路径解析的起始目录。

文件的组织：

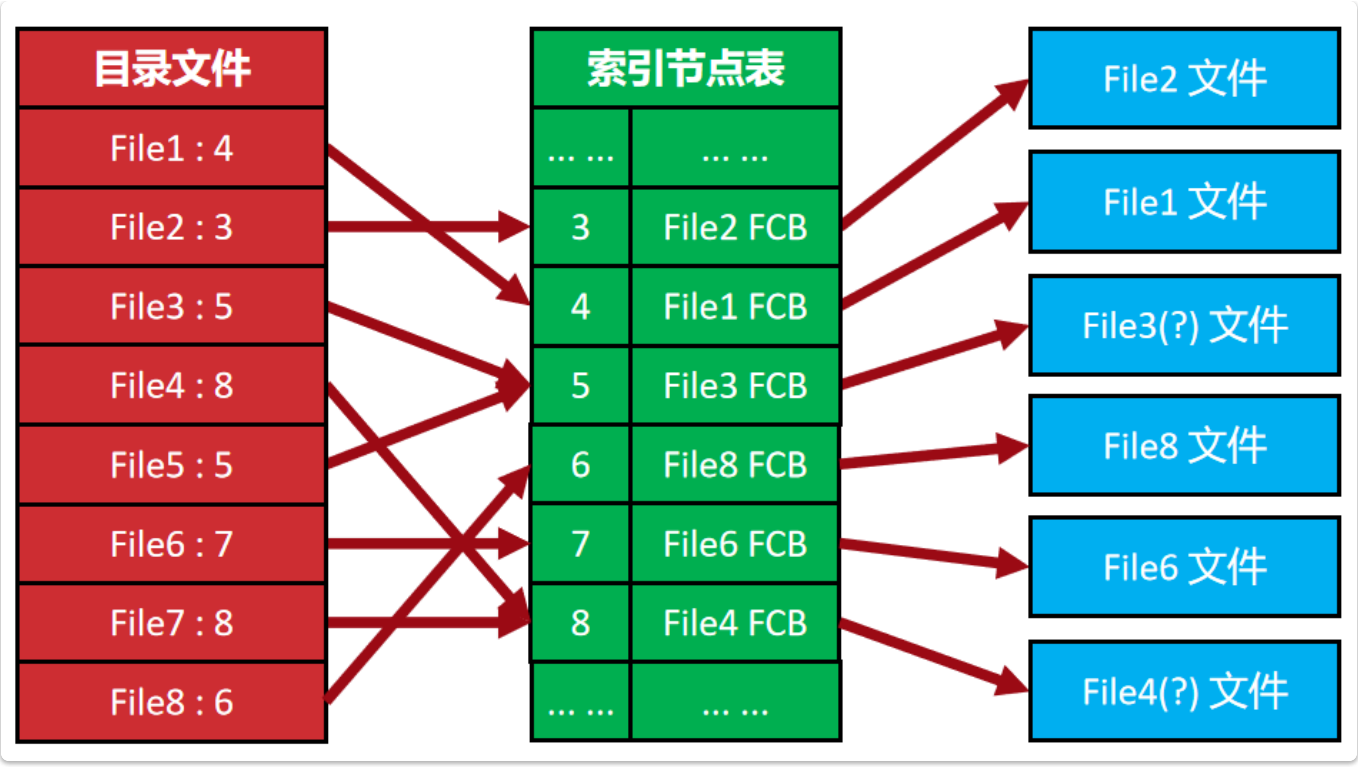
文件的存储：连续分配、链接分配、扩展分配。
链表的改进：不将指针、扩展长度等管理结构和文件数据存储在一起，而是给它们单独的存储空间。（将这个管理结构备份）
索引分配法：给每个文件创建一个线性索引表，每个表项记载对应于该逻辑块的物理块。
多级索引：像组织页表那样，将多个索引表以层次的形式组织起来，每个层次负责翻译逻辑块号的一部分，最终得到物理块号。
混合索引：文件的索引采取多级方法进行，但索引的级数随着文件块号的增加而增加。文件越靠前的部分，索引的级别越少。
空闲块的组织：考虑创建和删除文件时，一次分配或者释放多个空闲块。这是它和常规文件不同的地方。

存储方式	优点	缺点
连续存储	简单好实现，空闲块及其范围一目了然。分配和释放可以照内存分配的办法操作。	空闲块列表需要额外的空间来存储，而且该列表的大小是固定的，必须能够放下所有的物理块。
链表存储	相对而言也比较简单，分配单个块很容易。	分配和释放大量块时候会导致大量磁盘读写；找连续的扩展很麻烦。
索引存储	在分配单个块较容易的基础上，分配和释放大量块比较高效，因为索引表可以一次填充很多。	分配单个块需要做多级指针追逐。此外，结构复杂，找连续的扩展时候仍然很麻烦。

分组索引法。
位图法：用一系列二进制位对应每个块。如果该块被分配出去，那么位图的对应位置就置1，否则置0。
目录文件：包含目录信息的特殊文件。它们储存的是其下属各文件的属性，也即其FCB内容。

索引节点

将FCB拆成两部分，文件名直接储存在目录文件中，而其它属性放在索引节点中。索引节点单独使用一个线性表存放，目录文件仅仅储存索引节点到索引节点编号的映射。



硬链接：同一个文件在文件系统层面拥有两个文件名，它们都指向相同的索引节点，因而可以通过两个文件名访问同样的内容
软链接：一个独立于其目标的的链接文件，该文件的内容是它指向的原文件的路径。

虚拟文件系统：

元文件系统 ： 文件系统的文件系统，又叫虚拟文件系统。系统中所有的文件系统的根目录作为一个子目录出现在这个文件系统中，这样就可以用一种统一的路径描述来访问各个文件系统，减轻了应用程序的负担。（盘符法、路径法）

六.外设：

操作系统开发者最应该关心的：机制和策略的抽象设计。
四种传输方式。
三种设备。
中断分发：在多核处理器中，中断控制器可以将设备的中断平均分配给每个CPU。
中断节流：限制设备产生的中断率，也即在一定时间内产生的中断上限数目。（100包一个中断）
带超时时间的中断节流：在包多时可以充分发挥中断节流的优势，在包少时又不至于增加过多的延迟。
驱动程序：直接控制设备的接口程序，简称驱动。驱动程序是设备依赖的，一旦更换设备，就需要更换驱动程序。
驱动程序的功能
查找设备 扫描并检测硬件改动，识别应当识别的设备。
初始化设备 初始化并对设备进行必要设置，注册中断ISR。
响应设备请求 响应设备的中断等，在设备运行中与设备交互。

响应软件请求 执行操作系统和应用程序下发的设备操作指示。
上半部与下半部
在设备中断中，一般将中断响应分为上半部和下半部。上半部运行在中断上下文中，下半部则是一个独立的内核线程，该线程被ISR解除阻塞。可以将紧急的放到上半部
如果对时延敏感，或不能被打断，放在上半部；其它都放在下半部。

接口的分类：

阻塞接口 当设备无法即时完成操作或返回消息时，在接口上请求I/O操作的线程将阻塞，直到设备返回数据。一定需要操作系统介入。

非阻塞接口 当设备无法即时完成I/O操作或返回消息时，该接口将立即返回并将当前设备状态报告给调用线程。线程可以以合适的间隔轮询此接口，直到获取到数据。不一定需要操作系统介入。

（1）为什么需要非阻塞接口？一切接口都阻塞不好吗？
（2）阻塞接口和非阻塞接口哪个效率高？
（3）设备出故障，无法完成操作，导致线程永久阻塞怎么办？
（1）试探是否有数据，或者轮番试探多个设备；（2）数据量小的设备阻塞效率高；数据量大则反之。（3）增加超时返回。

七.互斥、同步：

临界资源 ;不能同时被两个指令流使用的资源。
临界区 ： 访问临界资源的程序段，临界区不能并发执行。
互斥：临界资源不能并发使用的现象称为互斥。

Peterson算法。
会出现（1）死锁，（2）活锁或者（3）饥饿吗?
（1）不会出现死锁，因为turn保证了有一方的等待条件总是遭到破坏。
（2）不会出现活锁，因为双方都在等待时均不放弃自己的进入意图。
（3）不会出现饥饿，因为turn只代表发生竞争时的优先进入权。