期末简答题总结

卷1:

CPU与外设之间的数据传输的四种方式:

1. 直接传输方式:
   CPU认为外设*始终就绪*，可随时收发数据。
2. 轮询传输方式:
   CPU 定时或不断查询外设状态，当外设准备好（如数据准备好发送或接收）时，进行数据传输。
3. 中断传输方式:
   中断传送方式是当外设需要与CPU进行信息交换时，由*外设向CPU发出请求信号*，使CPU暂停正在执行的程序，转去执行数据的输入/输出操作（即中断处理），数据传送结束后，CPU再继续执行被暂停的程序
4. 直接内存访问（DMA）传输方式:
   外设利用专用的接口（DMA控制器）*直接与存储器进行高速数据传送，并不经过CPU*（CPU不参与数据传送工作），总线控制权不在CPU处，而由DMA 控制器控制。

过程调用的过程:

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.

缺页异常的处理过程:

发生缺页异常-->进行缺页异常处理,进入OS-->写回换出页-->读入换入页-->修改页表-->重新执行,PC不变.

4种页面替换算法:

LFD(最优替换算法):<离线算法,很难落地>

当每次替换时，都寻找当前页面中*在最远的未来才会再次使用的那个页面*，并替换掉它。特别地，若一个页不再使用，则其对应的未来可以看作无穷远，应被首先淘汰。
缺点: 贪心算法，它只考虑一个单一的局部的状态，并且认为每次做出局部最优选择就能得到整体最优的结果。

FIFO(先进先出法):

换那个已经驻留了最长时间的页.

LRU(最久未用法):

每次都驱逐最久没有用过的那个页.

~~LFU(最不频繁使用): ~~

选择那些（在某个时间段内）访问次数最少的页进行替换。

栈式结构:

对于某种根据某个参数k并在全集C中产生一个子集S的策略，设m<n，当k=m时选出的Sm总是k=n时选出的Sn的子集。又称包含性性质。
(在这里是指，物理页资源的增加只会导致更多的页被包含进内存,不会导致本来就有的那些页被逐出内存。)
LFD、LRU都含有"最"字，因此满足这个性质（类比于"班里的前10名必然包含班里的前5名"）；FIFO则没有这个性质。

混合索引的思想:

索引分配 : 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。

线程的调度:

固定优先级(FP):

所有线程按照事先给定的优先级排序运行。(分为抢占式和非抢占式)

先到先服务(FIFO):

非抢占式。
公平，简单。
对长作业有利，对短作业不利。
不会饥饿。

短作业优先(SJF):

所有任务按照其运行时间排序，运行时间越短的任务优先级越高，越优先得到CPU。调度可在任务结束时和任务提交时发生。(**抢占式**)
短作业有利，长作业不利。
会饥饿。

响应比高优先(HRRN):

响应比 : T/R=1+(W/R) ，W是等待时间，R是运行时间。 (非抢占式)
不会饥饿。

时间片轮转法（RR）:

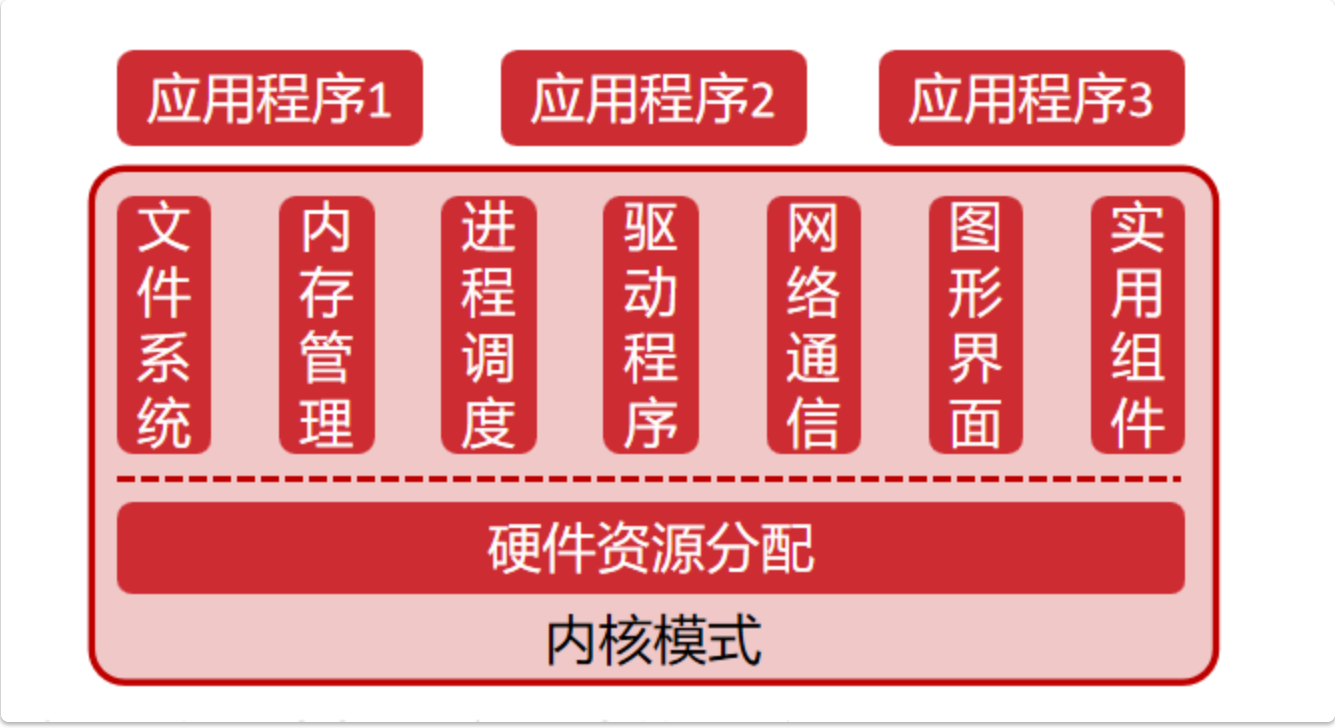
先来先执行，执行时间到了换下一个任务。
不会饥饿。

四种内核结构:

库结构:

无内核模式与用户模式的区分。
所有应用程序以及内核都在同一个保护域。
应用程序可以随时对任何资源做任何操作。
应用程序间为合作关系，操作系统的角色偏重协调而非管理。

宏内核结构:



有内核模式与用户模式的区分。
每个应用程序在不同的保护域。
内核的所有功能位于同一个保护域。
应用程序必须请求内核完成敏感资源操作。
应用程序间为合作或竞争关系，操作系统的协调和管理并重。

微内核结构:

内核中除了硬件资源分配只有进程调度和内存管理

有内核模式与用户模式的区分，每个应用程序在不同的保护域。
内核除基本功能外，其它功能分别位于不同的用户模式进程中。
应用程序必须请求守护进程中的策略分配敏感资源。
守护进程则转而使用内核提供的机制完成这些分配操作。

外核结构：

有内核模式与用户模式的区分，每个应用程序在不同的保护域。
内核仅负责硬件资源的安全分配与管理功能。
应用程序必须*自行*和被分配的硬件资源打交道完成功能。

卷2：

外设接口的三个寄存器：

数据寄存器，状态寄存器，命令寄存器

死锁的四个条件

互斥条件，持有条件（保持请求、无法剥夺），循环等待

进程与可执行文件

进程与可执行文件

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。

进程与线程：

进程与线程：一对一关系

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。作为特例，一个进程在创建时可以不包含线程，而是等待其他进程中的线程迁移过来。这在实现时间隔离的管程或服务器时非常有用。这一点我们在实时和混合关键度系统章节还要继续介绍。
关系	进程对线程可以为一对一、一对多、多对一、多对多关系。总的而言，至少在理论上讲，它们在数量上没有任何固定的对应关系。
一对一关系	最常见的关系，也是Linux在2.4版本之前的默认关系。在那和之前，Linux的线程和进程是一个东西，其task_struct里面同时含有线程的信息和地址空间的信息。 由于这种关系是如此普遍，因此很多书上会直接讲进程的调度、进程的状态。其中又以单指令流依附于单线程，单线程运行于单进程最为常见，因此很多人把进程、线程和指令流混为一谈也就不奇怪了。如果有人这样谈论概念，你要知道这是什么意思。 实际上，进程本身并不运行，运行的是它里面的线程上的指令流。

不只是一对一。