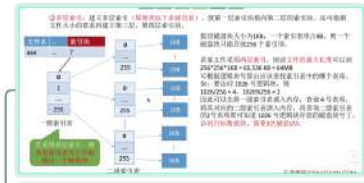
文件系统

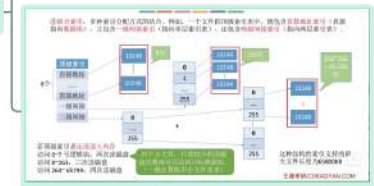
- 概览
  - 逻辑结构 — 文件内部组织方式
  - 目录结构 — 文件之间组织方式
  - 物理结构 — 文件在外存中的存储方式
  - 存储空间管理 — 外存中空闲块的管理
  - 操作系统需要提供其他文件管理功能
    - 文件共享
    - 文件保护
- 逻辑结构（针对有结构文件）
  - 顺序文件
    - 串结构 — 记录顺序和关键字无关
    - 顺序结构 — 记录顺序和关键字有关
    - 不方便增删记录
  - 索引文件 — 建立索引表，索引表本身可按照（不同）关键字排序
  - 索引顺序文件
    - 记录分组，每组对应一个索引表项
    - 先顺序查找索引表，找到分组，再顺序查找分组
- 目录结构
  - 将文件标识PCB组织起来就是文件目录
  - 目录结构
    - 单机目录
    - 两级目录
    - 多级树形目录
    - 无环图目录
      - 树形目录基础上增加一些指向同一结点的有向边，使得整个目录成为有向无环图
      - 为共享节点指定共享计数器，不同目录删除节点的时候计数器--，只有计数器==0时才真正删除文件
  - 索引节点
    - 目录项只包含文件名和索引节点指针
    - 索引节点包含除文件名以外其他文件属性
    - 目录项长度减少，每个磁盘块可以存放更多目录项，访问磁盘块的IO次数较少很多
- 物理结构
  - 顺序存储
  - 链接存储
    - 隐式 — 磁盘盘块包含指向下一个盘块的指针
      - 优点：方便文件扩展，无碎片
      - 缺点：只能顺序访问，指针需要额外存储空间
    - 显式 — 每个磁盘都有一张文件分配表FAT，需驻内存
      - 优点：支持随机访问，地址转换无需访问磁盘，访问效率提高
      - 缺点：FAT占用内存

物理块号	下一块
0	1
1	-1
2	5
3	-1
4	23
5	0
.....	
22	
23	3

FAT（文件分配表）

  - 索引存储 — 为文件数据块建立索引表，若索引表太大（大于一个磁盘块），可采用连接方案、多层索引、混合索引



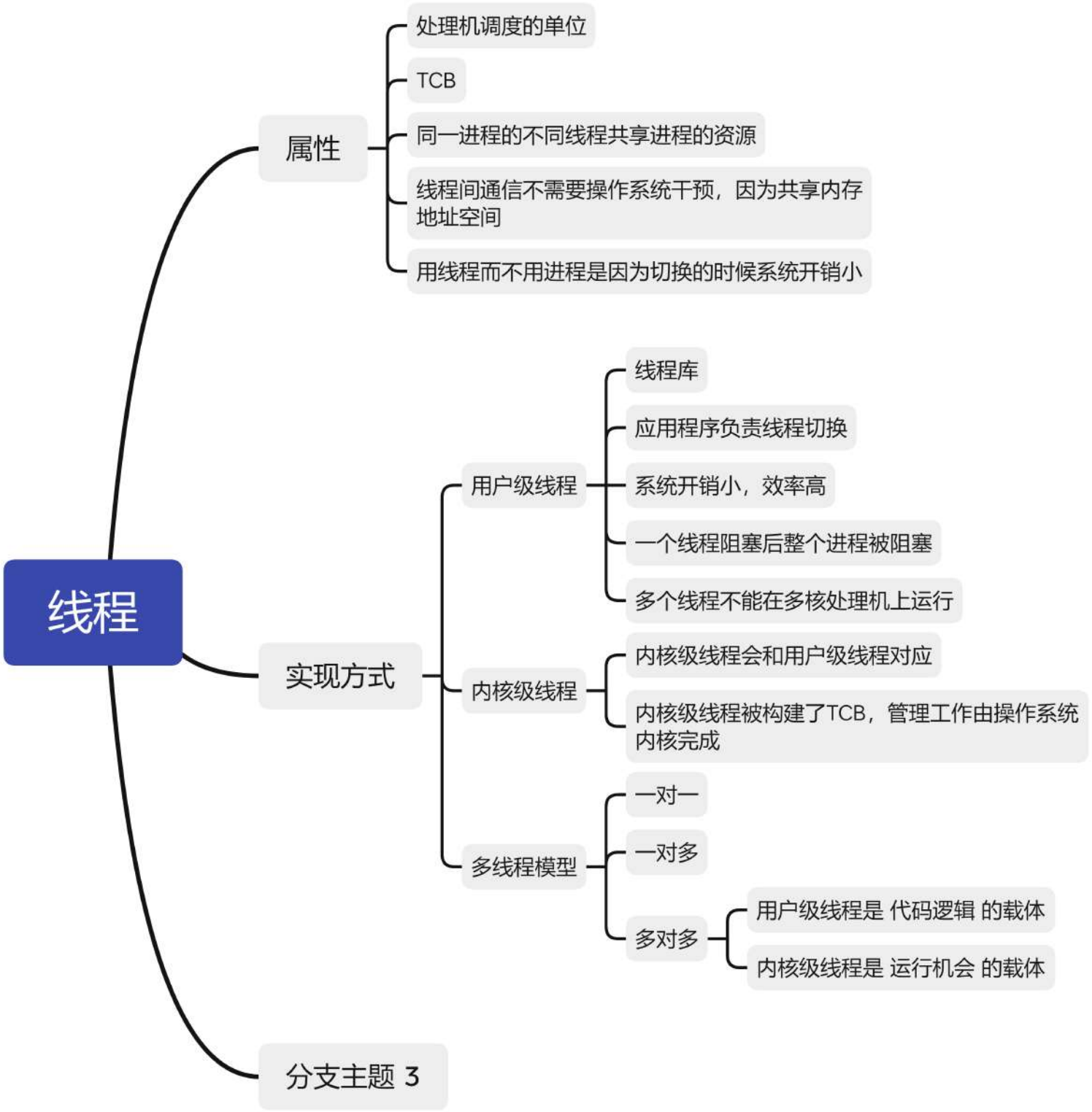


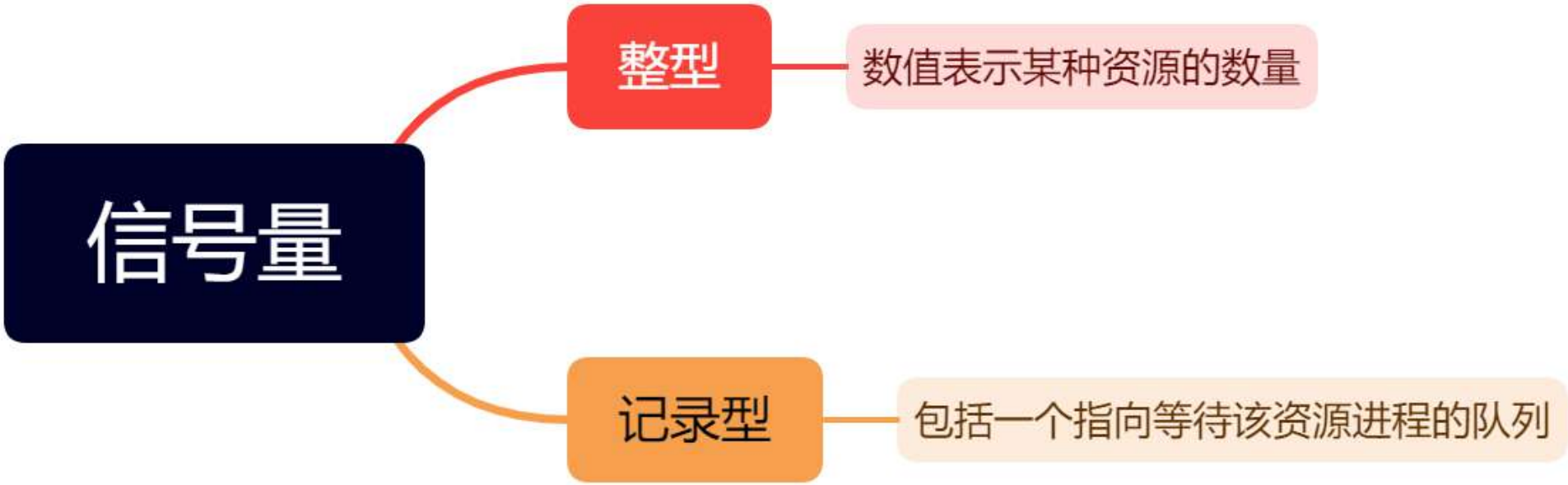
混合索引
优点：小文件减少访问次数

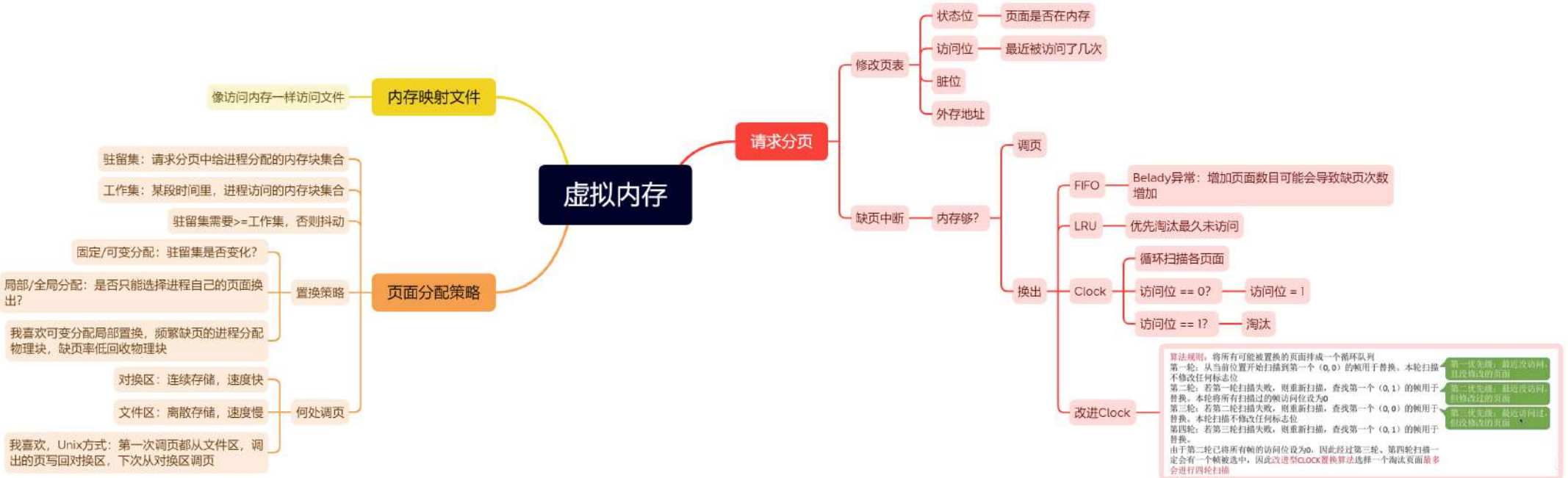
- 文件共享和保护
  - 共享
    - 同目录结构/无环图目录 — 硬链接
    - 快捷方式等符号链接 — 软链接
  - 保护
    - 口令存放在FCB中不安全 — 口令
    - 加解密需要时间 — 加密
    - 使用访问控制表实现 — 访问控制
- 文件基本操作

- 存储空间管理
  - 存储空间划分
    - 存放目录、空闲表等用于文件管理的数据 — 目录区
    - 文件区
  - 管理方法
    - 空闲表
      - 记录连续空闲区起始盘块号、盘块数
      - 首次适应、最佳适应、回收？
    - 空闲链表
      - 盘块链
        - 盘块为单位组成的链表
        - 链头取出空闲块 — 分配
        - 链到链尾 — 回收
      - 盘区链
    - 位示图
      - （列号，位号）或（字号，行号）和磁盘块一一对应

    - 成组链接法
      - Unix采用


线程
属性
处理机调度的单位
TCB
同一进程的不同线程共享进程的资源
线程间通信不需要操作系统干预，因为共享内存地址空间
用线程而不用进程是因为切换的时候系统开销小
实现方式
用户级线程
线程库
应用程序负责线程切换
系统开销小，效率高
一个线程阻塞后整个进程被阻塞
多个线程不能在多核处理机上运行
内核级线程
内核级线程会和用户级线程对应
内核级线程被构建了TCB，管理工作由操作系统内核完成
多线程模型
一对一
一对多
多对多
用户级线程是 代码逻辑 的载体
内核级线程是 运行机会 的载体
分支主题 3





操作系统

运行机制

操作系统的运行机制

- 简单了解程序的运行原理
  - 高级语言编写代码——>机器指令
  - 程序运行的过程就是CPU 执行指令的过程
- 两类程序
  - 内核程序
  - 应用程序
- 两类指令
  - 特权指令
  - 非特权指令
- 两种处理器状态
  - 内核态/核心态/管态
  - 用户态/目态
- 内核
  - 内核(Kernel)是操作系统最重要最核心的部分
  - 由很多内核程序组成操作系统内核
- 如何变态？
  - 内核态—>用户态 — 一条修改PSW的特权指令
  - 用户态—>内核态 — 由中断引起，硬件自动完成

only / only / only

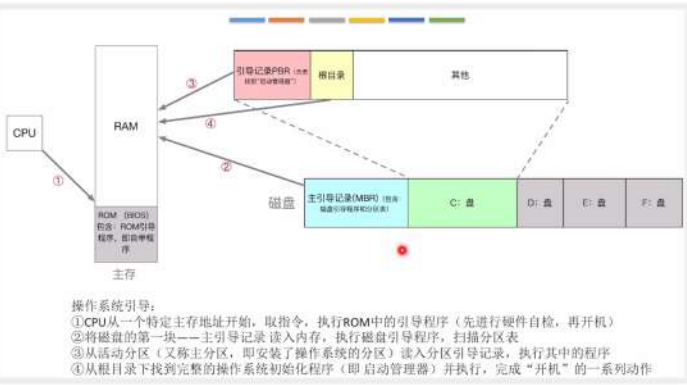
中断

- 内中断（异常，例外）
  - 陷阱 — 用户态执行特权指令
  - 故障 — 除数为0
  - 陷入指令 — 系统调用等
- 外中断
  - 时钟中断 — 每隔50ms切换程序运行，实现并发
  - I/O中断
- 处理
  - 通过'中断向量表' 找到中断处理程序
  - 中断向量表 — <中断信号类型，中断处理程序指针>构成的表

系统调用

- API是对系统调用的封装
- 什么功能要用系统调用实现？
  - 和共享资源有关的操作
  - 直接影响其他进程的操作

操作系统引导



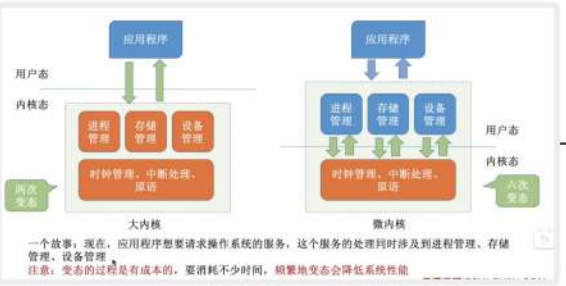
操作系统引导：
①CPU从一个特定主存地址开始，取指令，执行ROM中的引导程序（先进行硬件自检，再开机）
②将磁盘的第一块——主引导记录 读入内存，执行磁盘引导程序，扫描分区表
③从活动分区（又称主分区，即安装了操作系统的分区）读入分区引导记录，执行其中的程序
④从根目录下找到完整的操作系统初始化程序（即 启动管理器）并执行，完成“开机”的一系列动作

系统结构

虚拟机

- 最高特权级 — 直接运行在硬件之上
- 运行在Host OS上
  - 部分用户态，部分内核态
  - 系统调用转化为VMM对HostOS的系统调用

大内核和微内核结构

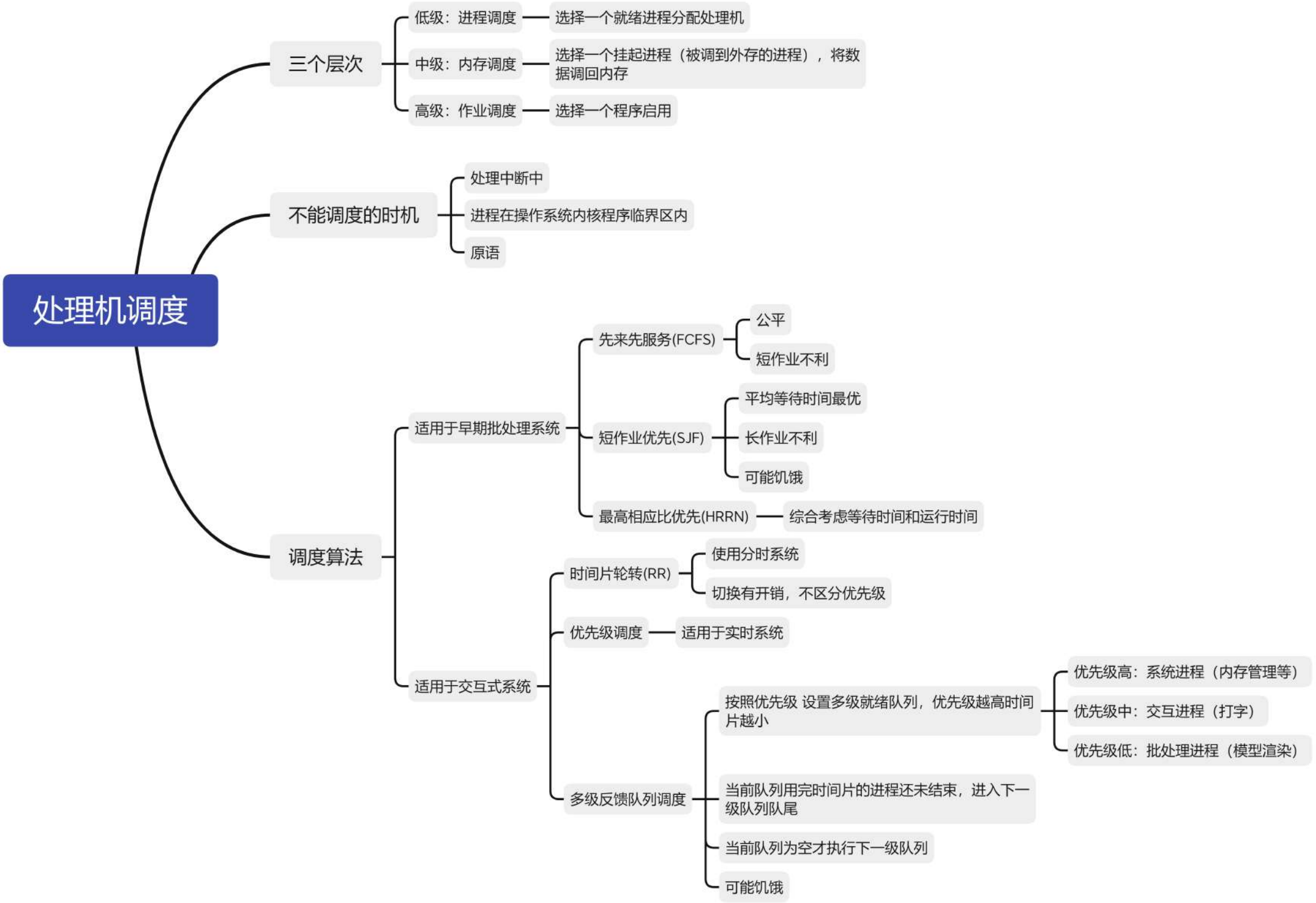


一个故事：现在，应用程序想要请求操作系统的服务，这个服务的处理同时涉及到进程管理、存储管理、设备管理
注意：变态的过程是有成本的，要消耗不少时间，频繁地变态会降低系统性能

- 保留几个最基本功能在内核 — 微内核特点
- 频繁变态 — 微内核缺点
- 结构清晰方便维护 — 微内核优点

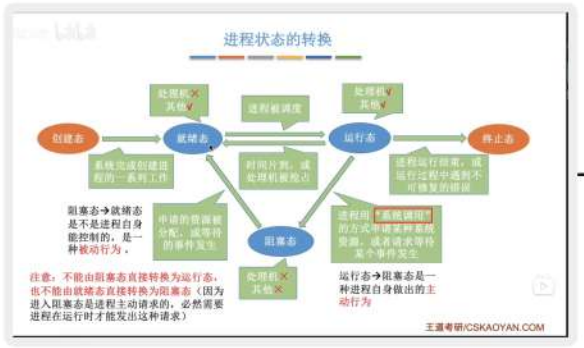
其他

	特性、思想	优点	缺点
分层结构	内核分多层，每层可单向调用更低一层提供的接口	1. 便于调试和验证，自顶向上逐层调试验证 2. 易扩充和易维护，向层之间调用接口清晰规范	1. 仅可调用相邻层，难以合理定义各层的边界 2. 效率低，不可跨层调用，各机间有执行时间长
模块化	将内核划分为多个模块，各模块之间相互协作。 内核 = 主模块+可加载内核模块 主模块：只负责核心功能，包括程序调度，内存管理 可加载内核模块：可以动态加载新模块到内核，而无需重新编译整个内核	1. 模块间逻辑清晰基于维护，确定模块间接口后即可各模块同时开发 2. 支持动态加载新的内核模块（如：安装设备驱动程序，安装新的文件系统模块到内核），增强OS适应性 3. 任何模块都可以直接调用其他模块，无需未然地处理调用进行通讯，效率高	1. 模块间的接口定义不必合理，实用 2. 模块间相互依赖，更难调试和验证
宏内核（大内核）	所有的系统功能都放在内核里（大内核结构的OS通常也采用了“模块化”的设计思想）	1. 性能高，内核内部各种功能都可以直接相互调用	1. 内核庞大功能复杂，难以维护 2. 大内核中某个功能模块出错，就可能导致整个系统崩溃
微内核	只把对硬件、保证、进程通信等最核心的功能放入内核，进程管理、文件管理、设备管理等功能以用户进程的形式运行在用户态	1. 内核小功能少，便于维护，内核可靠性高 2. 内核中的某个功能模块出错不会导致整个系统崩溃	1. 性能低，需要频繁的切换 用户态/核心态，用户态下的各功能模块不可以直接相互调用，只能通过内核的“消息传递”来间接通信 2. 用户态下的各功能模块不可以直接相互调用，只能通过内核的“消息传递”来间接通信
外核（exokernel）	内核负责进程调度、进程通信等功能，并将负责为用户进程分配未经抽象的硬件资源。且由外核负责保证资源使用安全	1. 外核可直接给用户进程分配“不虚拟、不抽象”的硬件资源，使用户进程可以更灵活的使用硬件资源 2. 减少了虚拟硬件资源的“映射层”，提升效率	1. 降低了系统的一致性 2. 使系统变得更复杂

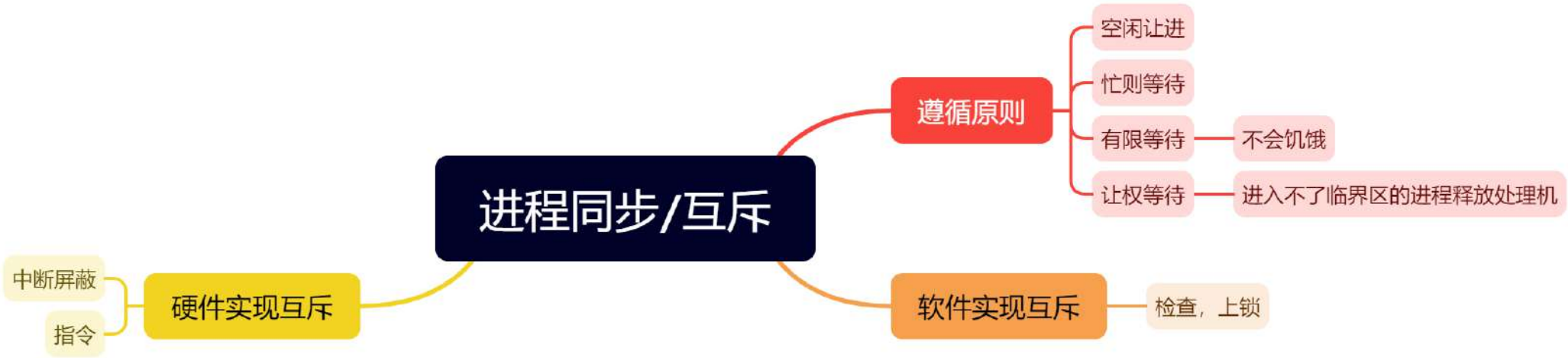

处理机调度
三个层次
低级：进程调度
选择一个就绪进程分配处理机
中级：内存调度
选择一个挂起进程（被调到外存的进程），将数据调回内存
高级：作业调度
选择一个程序启用
不能调度的时机
处理中断中
进程在操作系统内核程序临界区内
原语
调度算法
适用于早期批处理系统
先来先服务(FCFS)
公平
短作业不利
短作业优先(SJF)
平均等待时间最优
长作业不利
可能饥饿
最高相应比优先(HRRN)
综合考虑等待时间和运行时间
适用于交互式系统
时间片轮转(RR)
使用分时系统
切换有开销，不区分优先级
优先级调度
适用于实时系统
多级反馈队列调度
按照优先级 设置多级就绪队列，优先级越高时间片越小
优先级高：系统进程（内存管理等）
优先级中：交互进程（打字）
优先级低：批处理进程（模型渲染）
当前队列用完时间片的进程还未结束，进入下一级队列队尾
当前队列为空才执行下一级队列
可能饥饿

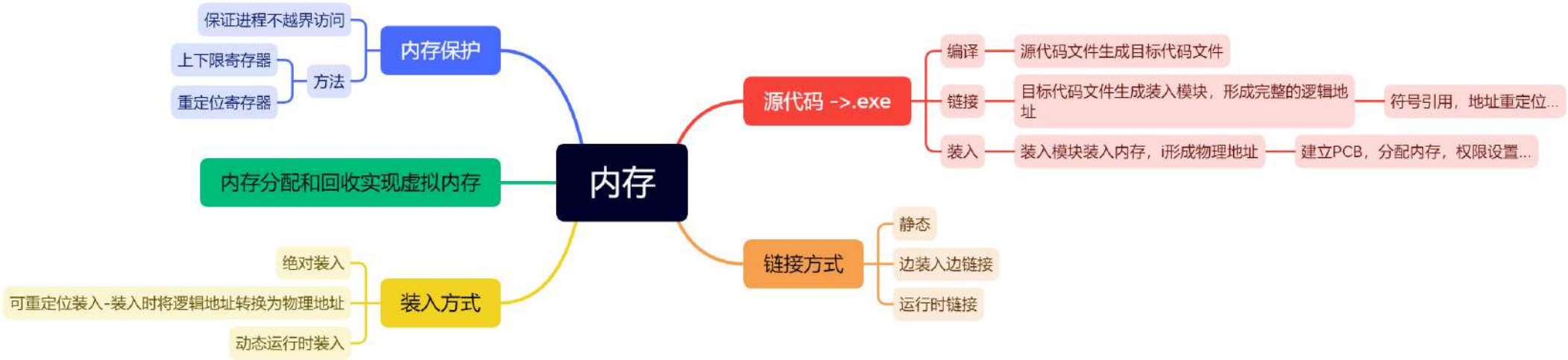
进程

- 是一种运行过程，是系统进行资源分配和调度的独立单位
- 组成
  - PCB（进程存在的唯一标识）
    - 进程 描述信息
    - 进程 控制管理信息
    - 资源分配清单
    - 处理机相关信息
  - 程序段 — 指令序列
  - 数据段
- 特性
  - 动态性（最基本特性）
  - 并发
  - 独立
  - 异步
  - 结构
- 状态
  - 状态转换条件



- 进程控制
  - 实现方式
    - 原语（关/开中断，一气呵成的程序）
  - 过程
    - 更新PCB信息
    - 将PCB插入合适队列
    - 分配/回收资源
- 进程通信
  - 共享内存
    - 增加 页表项/段表项 将同一片共享内存区映射到各个进程的地址空间中
    - 互斥访问
    - 高级通信：基于存储区
    - 低级通信：基于数据结构
  - 消息传递
    - 发消息/接收消息 原语实现
    - ⑥ 格式化的消息
    - 两种方式
      - 直接 — 点名道姓
      - 间接 — 发送到某个信箱
  - 管道
    - 管道文件（pipe），本质上是内存缓冲区
    - 更加简单，安全和可靠
- 组织方式
  - 链接
    - 按照进程状态将PCB分为多个队列
    - 操作系统持有指向各个队列的指针
  - 按照进程状态，建立索引表
  - 操作系统持有指向各个索引表的指针





内存分配

覆盖交换

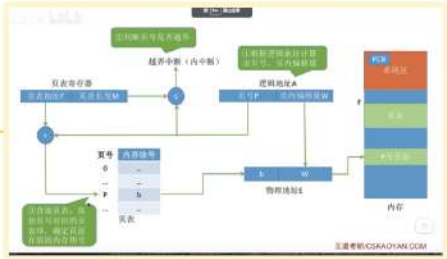
- 覆盖 — 单进程的程序段在覆盖区调入调出
- 交换 — 不同作业之间占有/空出内存

连续分配（进程内存连续）

- 单一连续分配 — 单道程序
- 固定分区分配 — 分区大小固定进行多道作业
- 动态分区分配 — 根据进程大小建立分区 — 算法
  - 首次适应（从头到尾找适合分区） — 开销小
  - 最佳适应（优先用小分区） — 满足大进程需求
  - 最坏适应（优先用大分区） — 减少难以利用的小碎片
  - 临近适应（同首次适应但从上次查找结束位置开始查找）

分页（放在不连续的页里）

- 概念
  - 页框 = 页帧 = 内存块 = 物理块 = 物理页面（物理空间）
  - 页面 = 页（逻辑空间）
- 页表
  - 记录页和页框的关系
  - 和进程绑定，页号是隐含的
  - 逻辑地址 = <页号，页内偏移>
- 地址变换
  - 逻辑地址计算出【页号】【页内偏移】
  - 页号 < 页表长度
  - 页表项 = 页表起始地址 + 页号 * 页大小
  - 页表项记录内存块号 & 页内偏移 ——》物理地址



- 页表一般存放在连续内存中
- 需要的页表分级数 = [逻辑地址页号长度 / 单个页表可存放页表项数目的幂] +1

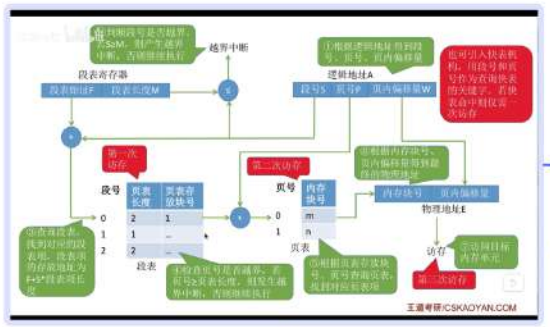
1. 若采用多级页表机制，则各级页表的大小不能超过一个页面

例：某系统按字节编址，采用 40 位逻辑地址，页面大小为 4KB，页表项大小为 4B，假设采用纯页式存储，则要采用（）级页表，页内偏移量为（）位？

页面大小 = 4KB =2^{12}B，按字节编址，因此页内偏移量为12位
页号 = 40 - 12 = 28 位
页面大小 = 2^{12}B，页表项大小 = 4B，则每个页面可存放 2^{12} / 4 = 2^{10} 个页表项
因此各级页表最多包含 2^{10} 个页表项，需要 10 位二进制位才能映射到 2^{10} 个页表项，因此每一级的页表对应页号应为10位，总共28位的页号至少要分为三级

逻辑地址：页号 28位 | 页内偏移量 12位

段页式



逻辑地址 = <段号，页号，页内偏移>

分段

逻辑地址 = <段号，段内地址>