操作系统复习

Chapter 1 导论

计算机系统组成

- 冯诺依曼体系结构
  - 五大部件：运算器、控制器、存储器、I/O
  - 存储器与CPU分离：指令存储与数据存储共享存储器
- System boot：系统引导
  - linux启动：ROM BIOS（基本输入输出系统）
  - MBR：主引导记录，是启动操作系统的起始部分代码
- BIOS启动过程
  - 开机自检
  - 初始化硬件设备
  - 找到一个启动的操作系统
  - 把操作系统的其实部分复制到特定位置，并跳转到该位置
- 现代计算机是中断驱动的，因此启动后在等待中断
- 中断过程：
  - 停止当前的任务，保存当前地址，保存上下文（Linux保存在内核态栈中）
  - 禁用中断以防止丢失中断
  - 将控制权转交ISR：中断服务例程（存储在固定位置的程序，可以唤起中断处理）和中断向量表
  - 处理完毕后返回程序
- 在保护模式下，OS填写IDT（中断描述符表），CPU根据中断控制器中获取的中断向量号跳转到ISR入口地址
  - 保存上下文
  - 处理中断
  - 恢复上下文

I/O结构

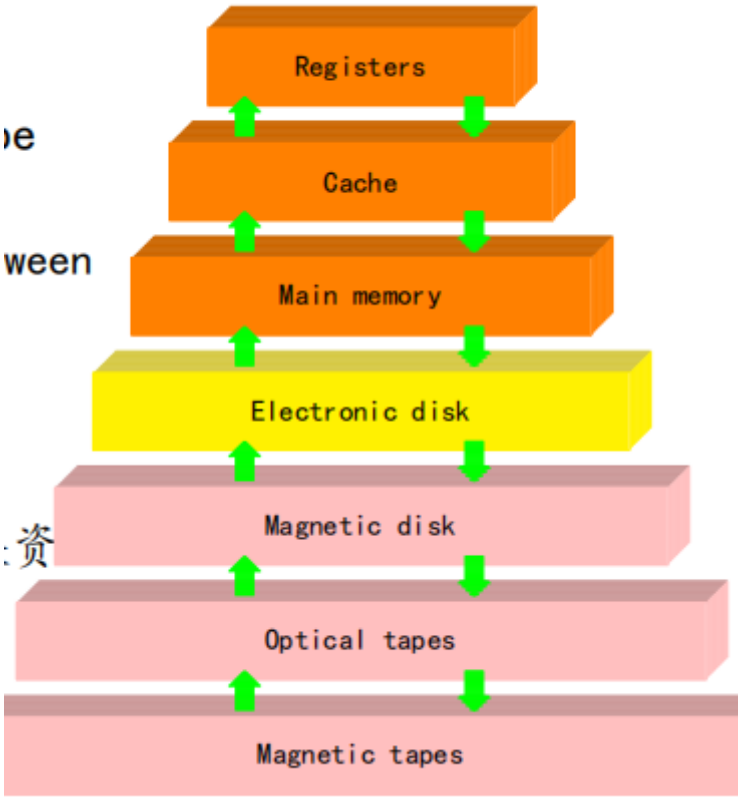
- I/O拥有自己的寄存器或缓冲，可以和CPU并发执行
  - 共享或竞争存储周期
  - 内存控制器
- 同步I/O：
  - 优点：始终确定哪一个设备正在中断
  - 缺点：不能处理并发I/O，在I/O上浪费过多计算资源
- 异步I/O：需要跟踪许多I/O请求
  - 设备状态表
  - 设备的等待队列
  - 中断发生时，操作系统进入设备表来确定I/O状态，并维护设备表以响应I/O
  - 优点：提高系统运行效率
- DMA：直接内存访问
  - 在主存和缓冲中快速直接地搬运数据而无需CPU干预，每块只产生一个终端

- 抢占存储周期

存储结构

- 存储评判的关键：seek time
- 内存（Main Memory）是RAM，CPU可以直接访问

  外存（Secondary Memory）是磁盘等，是内存的扩充
- 存储层次



- 高速缓存技术
  - 寄存器看做内存的cache
  - 内存看做外存的cache
- 内存墙：CPU速度增长远远快于CPU芯片外存储速度增长

硬件保护

- 硬件提供的四种保护
  - 双模式操作：用户模式下权力受限
    - 启动时从特权模式开始
    - 用户程序运行在用户模式
    - 中断和操作系统运行在特权模式
    - 设置定时器、清空内存、开关中断、维护设备状态表，访问I/O设备均为特权模式
    - 启动、系统调用、异常、中断下计算机运行在管理模式
  - I/O保护
    - 所有I/O指令全部为特权指令，用户模式需要使用系统调用来访问
  - 存储保护
    - 基址寄存器+限制寄存器保护
    - 操作系统可以任意访问内存

- CPU保护
  - 使用定时器来解决死循环问题
  - 设置一个特定运行时间，当计时器递减为负时，操作系统接管

系统结构

- 多任务
- 时间共享
- 操作系统
  - 控制软硬件
  - 处理特权指令
  - 系统调用
    - 陷入特定位置
    - 将控制权交给操作系统的特定部分，并设置监管者模式
    - 处理系统调用
    - 将控制权交还给系统调用的下一条指令

计算环境

- 传统计算、移动计算、分布计算、客户机——服务器计算、对等计算、虚拟化、云计算

操作系统的发展

- 动力
  - 不断提高计算机资源利用率的需要
  - 方便用户
  - 器件的不断更新换代
  - 计算机体系结构不断发展
- 历程

  无操作系统——批处理系统——分时系统——实时系统——PC——分布式和并行凶——嵌入——移动系统

批处理系统（Batch system）

- Batch的含义：供一次加载的磁盘或硬盘，通常由多个作业组装而成
- 单道批处理：先入先出，同一时刻只用一个作业在内存中
  - 批处理系统引入的目的：提高系统资源利用率
  - 特征：自动性、顺序性、单道性
  - 问题：速度较慢
    - CPU速度和I/O速度之间的矛盾（目的）：引入脱机I/O，使用低速外围机（方法），程序和数据都脱离主机（内涵）
    - CPU利用率较低：引入磁盘和多道程序
- 多道批处理：系统中同时驻留多个作业，**共享内存、复用CPU**
  - 优点：
    - 提高CPU利用率
    - 提高内存和I/O设备利用率

        - 提高系统吞吐量
    - 缺点：
        - 无用户互动
        - 作业时间过长
    - 特征：多道性、无序性、调度性（作业、进程调度）
    - 对操作系统的需求：
        - 作业调度
        - 内存管理
        - CPU调度
        - I/O支持
        - 设备分配

分时操作系统

- 工作方式：一台主机链接多个终端，用时间片轮转方式处理服务请求
- 关键技术：
    - 及时接收
    - 及时处理：交互作业始终在呢村中、时间片轮转
- 特征：多路性、独立性、及时性、交互性
- 影响响应时间的因素：
    - 机器处理能力
    - 请求服务的时间长短
    - 链接的终端数目
    - 服务请求分布
    - 调度算法
- 分时与批处理结合：分时有限、批处理在后

实时系统

- 定义：能响应外部事件请求，在规定的严格时间内完成对该事件的处理，并控制所有实时设备和实时任务协调一致工作
- 特征：
    - 专用系统
    - 实时控制
    - 高可靠性
    - 事件驱动和队列驱动

操作系统概述

- 操作系统的角色
    - 用户与计算机硬件之间的接口
        - 命令接口、图形用户接口
        - 编程接口
    - 计算机资源的管理者
        - 处理器、存储器、I/O设备、文件
        - 扩充机器（或虚拟机）

- 操作系统的定义：一组控制和管理计算机软硬件资源、合理地对各类作业进行地阿杜以及方便用户的程序的集合
- 操作系统的结构
  - Layer层次模型（经典模型）
    - 最高层：接口（用户），中间层：对对象进行操作管理的软件集合，最底层：硬件抽象层
    - 每层只能利用更底层的功能，效率较差，其难点在于合理定义各层
    - 举例：类UNIX和早期Windows
  - 微内核
- 设计原则：
  - 方便性（最重要）
  - 有效性在（最重要）
  - 可扩容性
  - 开放性

Chapter 2 操作系统结构

操作系统组成

- 进程管理：
  - 进程控制：创建、销毁、挂起、恢复进程，由进程控制原语完成
  - 进程同步：互斥（同步）、死锁避免、预防、检测消除
  - 进程通信：包括直接、间接通信
  - 作业、进程调度
- 存储管理：
  - 目的：方便用户使用、提高存储器利用率
  - 内存分配：动态、静态
  - 内存保护：设置上下界寄存器
  - 内存映射：逻辑地址和物理地址
  - 内存扩充：虚拟存储技术
- I/O系统管理
  - 目的：提高利用率和速度，方便用户
  - 缓冲罐里：解决CPU-I/O矛盾
  - 设备分配：设备、设备控制器、I/O通道的分配和回首
  - 驱动：读写、中断、构成通道程序
  - 设备独立性和虚拟设备：program与具体设备无关，易于重定向，增加可移植性
- 文件管理
  - 目的：方便用户，提供安全性
  - 文件存储空间管理
  - 目录管理
  - 读写管理和存取控制
- 辅存管理
  - 重要性：计算机操作地性能受限于磁盘系统调度和算法
  - 空闲空间管理
  - 存储分配
  - 磁盘调度
- 用户接口和命令解释系统

    - 用户接口
        - 命令接口
        - 程序接口：系统调用、库函数
        - 图形接口
    - 命令解释接口
        - 用户和操作系统的接口，允许用户直接输入命令并进行解释，位于内核或特定程序中
- 保护和安全：允许或拒绝访问某些资源

操作系统服务

- 装载并运行程序、提供I/O服务，提供文件系统和文件操作、提供通信服务、提供差错检测服务、管理资源分配、统计资源使用、保护系统
- 提供服务最基本方式：系统调用

系统调用

- 应用编程接口（API）为程序员提供了调用实际的系统调用
- 系统调用因计算机不同而不同
- 传参方式：
    - 寄存器传参
    - 块、表传参，块地址在寄存器中（Linux）
    - 栈传参
- 类型：
    - 进程控制类
    - 文件管理类
    - 设备管理类
    - 通信类
        - 包括消息传递模型和共享内存模型
    - 信息维护类

系统程序

- 为程序开发和执行提供了一个方便的环境
- 文件管理、状态信息、文件修改、程序语言支持、程序加载与执行、通信、后台服务

操作系统的特征

- 并发：（最基本）
    - **并行是多个事件同时刻发生，并发是多个事件在同一时间间隔内发生**
- 共享：（最基本）
    - 系统中的资源可供内存中多个并发执行的进程共同使用
    - 互斥共享和同时访问
- 虚拟：
    - 通过某种技术把一个物理试题变为若干个逻辑上的对应物
    - 若n是某一物理设备所对应的虚拟逻辑设备数，则虚拟设备速度必然是物理设备速度的1/n
- 异步：运行金服不可预知

操作系统的抽象模型和体系结构

- 进程模型
  - 进程地址空间
  - 进程描述符和上下文：程序指针、堆栈指针和寄存器
  - 进程调度
  - 通信机制：信号、信号量、管道、消息队列、套接字
- 线程模型
  - 线程是指令在进程地址空间的执行轨迹，可以单线程或多线程
  - 任何一个线程都属于某个进程
- 操作系统结构
  - 简单结构：功能模块和用户应用混杂在同一地址空间，模块之间可以互相调用
  - 单一内核结构：用户应用只能通过中断、异常、系统调用方式使用操作系统服务
    - 缺点：难以执行和维护
  - 模块化结构：
    - 模块之间定义了函数调用类型的接口
    - 提高了操作系统的可维护性
  - 层次结构
    - 每个层次只能用低层次的操作，且把本层服务提供给高层次
    - 最高层为用户，中间层为操作管理集合，底层为硬件服务
    - 优点：简化了系统调试和验证

      缺点：难以合理定义各层、效率较低
  - 微内核
    - 例子：鸿蒙、早期Windows
    - 将非必要组成部分移出内核，只保留基本的功能
    - 功能：最小化的进程和内存管理、为用户和客户端提供消息传递
    - 优点：可扩展性、可移植性、安全性和可靠性

      缺点：由于功能开销，微内核性能会受损
  - 混合内核
    - 扩大微内核结构，把关键的服务程序和驱动程序加入到内核
    - 削弱了可扩展性、灵活性和可靠性的优点
- 虚拟机
  - 操作系统为多进程创造虚拟机，每个虚拟机运行在自己的进程和虚拟内存中
  - 优点：通过独立性进行保护、每个虚拟机中都可以运行不同的操作系统
- 机制与策略相分离
  - 机制决定了如何来做
  - 策略决定了做什么

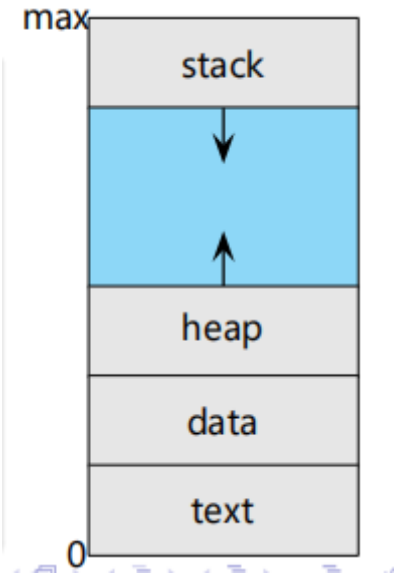
Chapter 3 进程

程序的并发

- 程序并发的特征
  - 间断性：并发程序“执行——暂停执行——执行”
  - 失去封闭性：由于资源共享，程序间可能相互影响
  - 不可再现性：程序间相互影响（对“顺序”的否定）
- 程序并发的条件：应当避免不可再现性
  - 当一个程序在写内存某位置时，其他程序不能读、写这个位置
  - 当一个程序在读内存某位置时，其他程序不能写这个位置

进程的概念

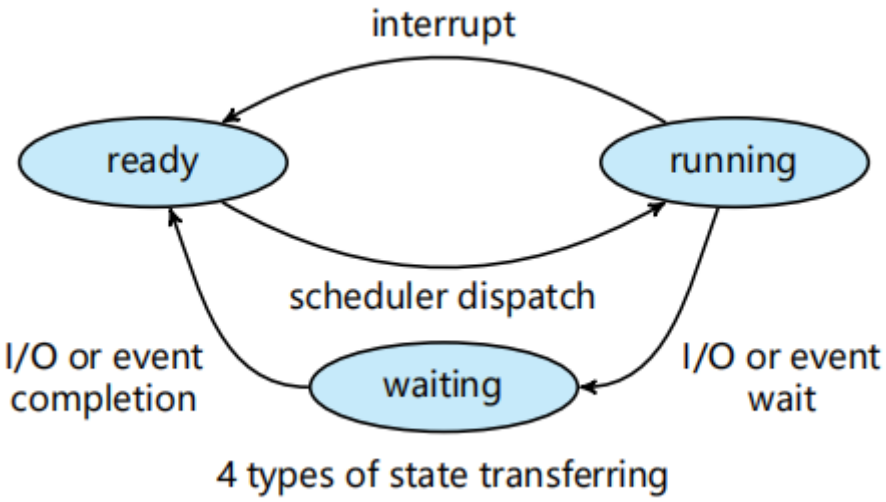
- 进程的内容：
  - 代码段、程序计数器和寄存器、堆栈、全局数据段、堆



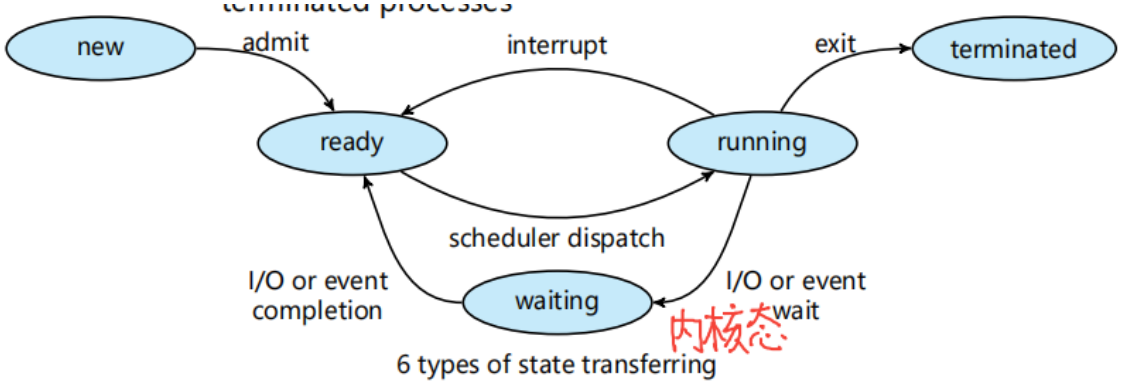
- 进程的特征
  - 动态性：（最基本）
    - 具有生命期，因创建而产生，因调度而执行，因得不到资源暂停执行，因销毁而消亡
  - 并发性：
    - 多道
  - 独立性：
    - 进程是一个能够独立运行的基本单位，也是系统中独立获得资源和独立调度的基本单位
  - 异步性：
    - 进程各自独立地以不同速度推进，导致不可再现
    - 操作系统必须采取措施来保证程序之间协调运行
  - 结构特征：
    - 进程映像 = 程序段 + 数据段 + 进程控制块

进程状态

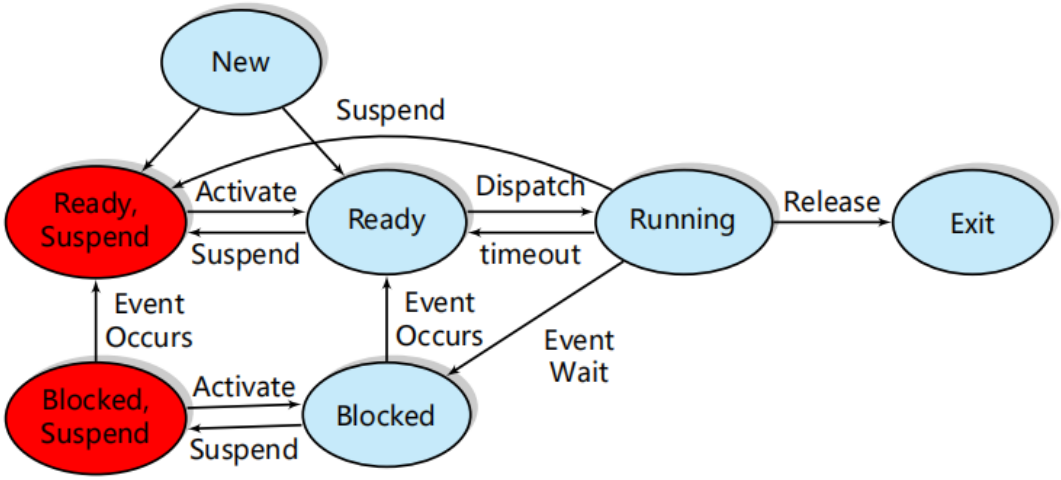
- 三状态模型：



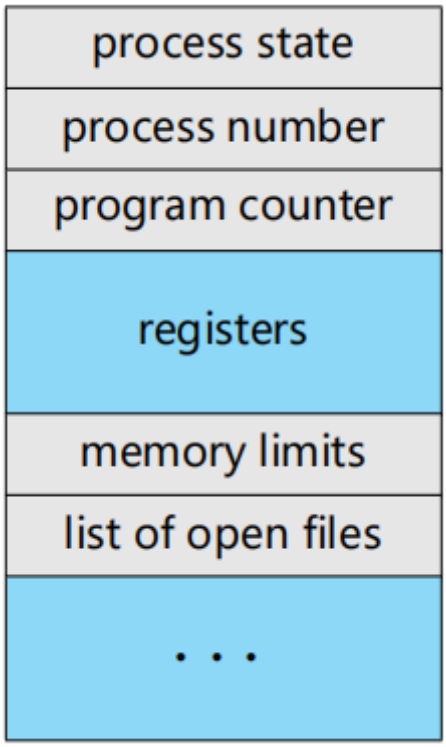
  - 就绪：存在就绪队列
  - 执行
  - 等待：存在等待队列，在等待I/O或其他资源
- 五状态模型



  - 新状态：被创建和预分配资源
  - 终止状态：从就绪队列移除但未被销毁，其他进程可以从这里得到一些信息
- 七状态模型

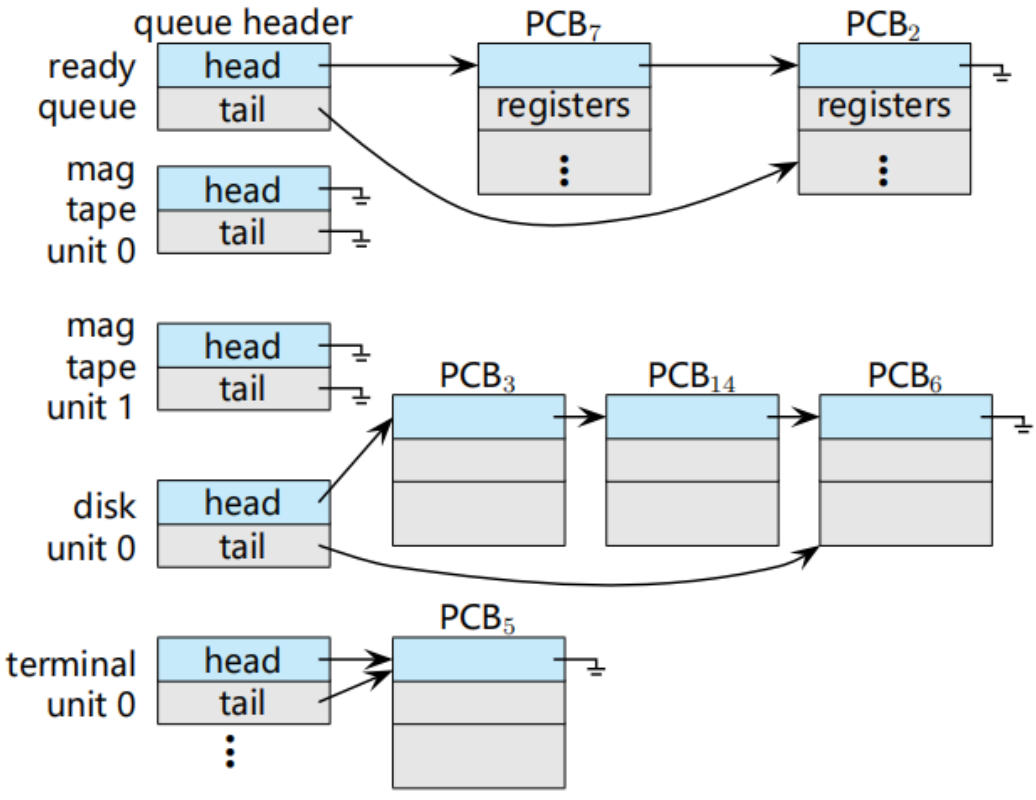


  - 挂起类状态
    - 挂起后处于静止状态：静止就绪，静止阻塞
    - 非挂起的活动状态：活动就绪、活动阻塞、执行
- 进程控制块（PCB）

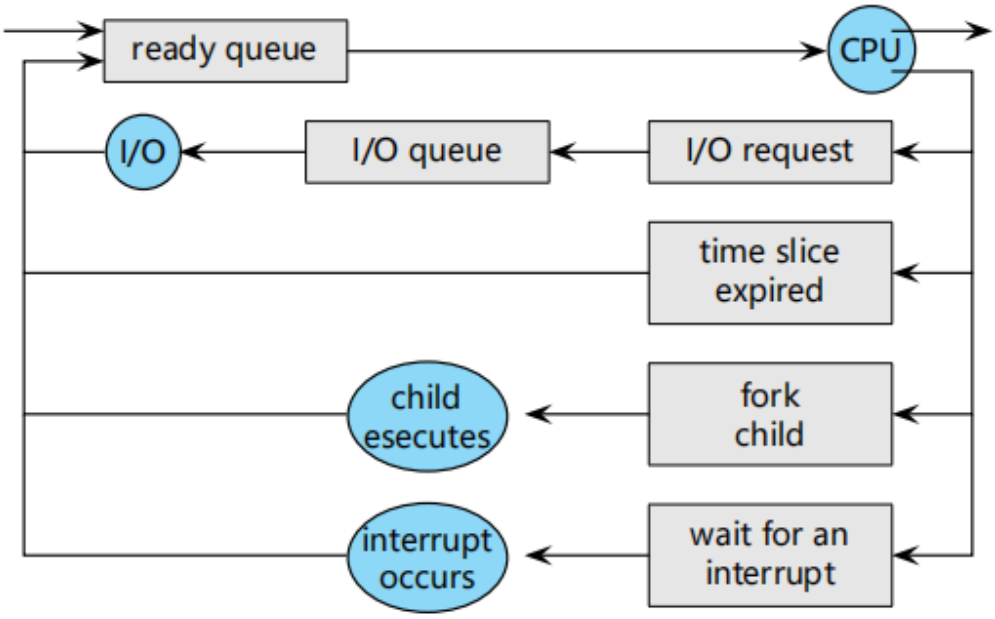


进程调度

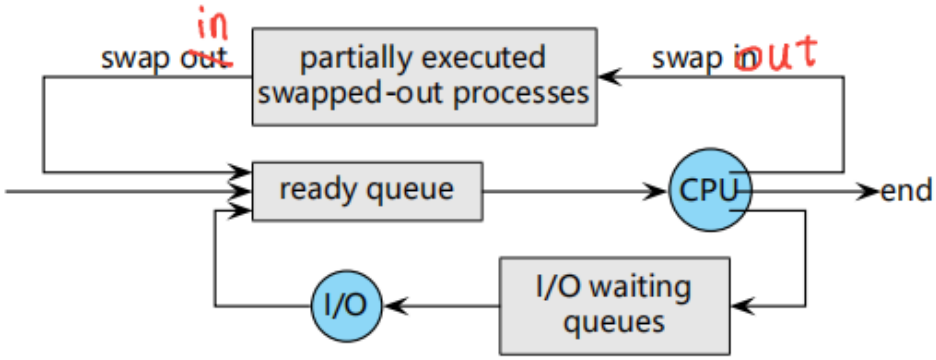
- 调度队列
  - 作业队列：进程刚进入系统时进入作业队列
  - 就绪队列：内存中就绪的多进程保存在就绪队列
  - 设备队列：进程需要等待I/O，则被移动到这个设备磁盘的设备队列上



  - 队列的转化



- 调度程序
  - 长期调度：从作业缓冲池中选取进程放入内存中的就绪队列
    - 执行频率：很低，分钟级别
    - 控制多道程序度：内存中进程的数目
    - 只有进程离开系统时，才需要长期调度
    - 需要仔细选择I/O密集型和CPU密集型进程
  - 短期调度：从就绪队列中选取合适的进程加载到CPU
    - 执行频率：很高，毫秒级别
  - 中期调度：将进程从内存或CPU竞争中移除
    - 功能：通过交换，**降低多道程序度**



- 上下文切换
  - 保存旧进程的寄存器、进程状态，加载新的进程状态
  - 上下文切换是纯额外开销，时间决定于：
    - CPU和存储的速度
    - 寄存器数量
    - 特殊指令

进程操作

- 进程创建：通过进程标识符pid来唯一区别进程
  - 资源共享
    - 父进程和子进程可能共享所有资源
    - 子进程只能共享父进程部分资源
    - 子进程不能共享资源
  - 执行
    - 父子并发执行
    - 父进程等待，直至某个子进程全部执行完毕
  - 地址空间
    - 子进程是父进程的一个完全复制品
    - 子进程加载了新程序
  - fork：
    - 调用n次，创建2^n-1个进程，共2^n个进程（无if）
    - 一般子进程pid = 0
- 进程终止
  - 自行终止：调用exit()，资源被操作系统回收
  - 其他进程终止：Win32使用TerminateProcess()终止其他进程
  - 用户终止：kill()
  - 父进程终止：
    - 子进程使用了超过它分配的资源
    - 分配给子进程的任务不再需要
    - 父进程正在退出

    若父进程终止，有些操作系统下子进程也应当终止——**级联终止**

    若父进程未调用wait，则需要将子进程托管到根进程

进程间通信（IPC）

- IPC的优点
  - 信息共享
  - 计算加速
  - 模块化
  - 方便
- 共享内存模型
  - 生产者——消费者问题
    - 需要循环队列，充当信息缓冲，分为无界缓冲区和有界缓冲区
    - 生产者在队满时等待，消费者在队空时等待
- 消息传递模型
  - 直接通信：每个进程要指定消息的发送者或接收者，或接收者不需要指发送者
  - 间接通信：标识一个邮箱
    - 只有两个进程共享一个邮箱时，才建立通信链路
    - 一个链路可以与两个或更多进程关联
    - 两个通信进程之间可能有不同链路，每个链路对应一个邮箱

关联同一邮箱的多个进程接收方案：

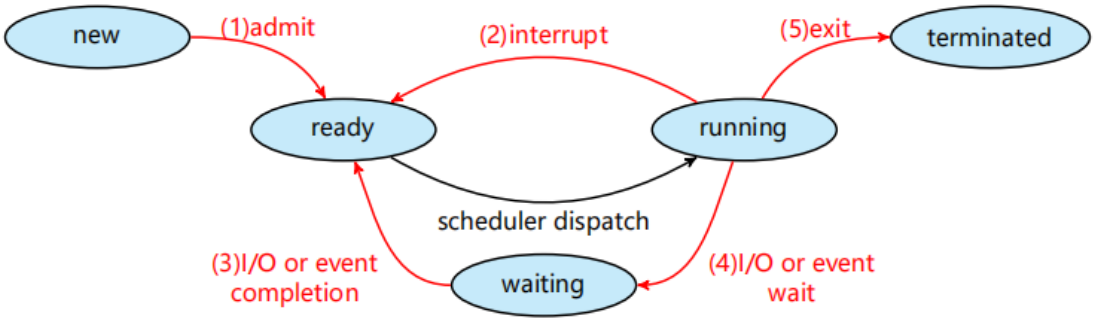
- 允许一个链路最多关联两个进程
- 允许一次最多一个进程执行操作receive()
- 系统随意选择一个接收进程，或让发送者指定接受者

- 同步：
  - 阻塞发送、接收：等待直到对方响应
  - 非阻塞发送、接收：不等待，可能接收到空消息
- 缓冲
  - 零容量：链路中不能有消息在等待，发送者必须阻塞
  - 有限容量：链路满时发送者必须阻塞
  - 无限容量：发送者从不阻塞

Chapter 4 CPU调度

基本概念

- CPU调度的情况
  - 新建切换到就绪：可以不调度
  - 运行切换到等待：必须CPU调度
  - 运行切换到就绪：可以不调度
  - 等待切换到就绪：可以不调度
  - 终止：必须CPU调度



- 抢占式调度：如果进程更新数据时被其他进程抢占，同时其他进程试图读数据，可能会数据不一致
  - 需要特定硬件，如计时器
  - 对共享数据造成额外同步开销
  - 可以采用关中断或同步手段来避免内核被中断
- 调度程序（dispatcher）
  - 切换上下文
  - 切换到用户模式
  - 跳转到用户程序合适位置，启动程序

  调度延迟：停止一个进程到启动另一个进程所需时间，越短越好

调度准则

- CPU利用率
- 吞吐率：一个时间单元内进程完成的数量
- 周转时间：所有时间之和，包括I/O时间、等待时间等
- 等待时间：就绪队列中等待时间之和
- 响应时间：从提交请求到产生第一响应的时间

调度算法

- FCFS：等待长进程释放CPU（护航效应），非抢占
- SJF短作业优先：常用于长期调度，抢占或非抢占，可能导致饥饿
  - 预测执行时间方法：指数平均，t_n为第n个CPU执行时间，τ_n为第n次预测值

    $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$

    当α趋近0，则更重视历史，而α趋近于1，则更重视最近
  - 最小平均等待时间
- 优先级调度：造成无穷阻塞和饥饿
  - 解决无穷阻塞和饥饿：老化，随时间递减优先级，直至为最高优先级
  - 优先级反转问题解决方案
    - 优先级继承：当高优先级进程等待低优先级进程占用的资源时，低优先级进程将暂时获得高优先级进程的优先级，释放共享资源后恢复
    - 设优先级上限：一旦某个进程获得资源，就将该进程优先级提升到所有可能访问到资源的进程中的最高优先级
- RR调度
  - 性能取决于时间片大小
    - 时间片太大，退化为FCFS
    - 时间片太短，调度开销太大
- 多级队列调度
  - 前台队列使用RR，后台队列使用FCFS
  - 队列间调度使用固定优先级抢占调度（前台队列绝对优先），或分时间片（前台分的多）
- 多级反馈队列
  - 如果一个进程使用了过多的CPU时间，它会被迁移到更低的优先级队列
  - 先给予一个时间片，如果完不成，就向下调度

多处理器调度

- 方法
  - 非对称多处理：一个处理器处理所有调度决定
  - 对称多处理：自我调度
- 处理器亲和性
  - 软亲和性：操作系统试图保持进程在统一处理器上
  - 硬亲和性：使某个进程运行在某个处理器子集上
- 负载平衡
  - 推迁移：特定的任务周期性检查每个处理器负载，如果发现过载，则推到空闲处理器
  - 拉负载：一个空闲处理器从忙处理器上拉走一个等待任务
- 多处理器
  - 内存停顿：当一个处理器访问内存时，它花费大量时间等待所需数据

实时CPU调度

- 硬实时系统：需要严格地在一个能够保证的时间内完成一个重要的任务
- 软实时系统：稚嫩保证关键任务的进程会优先于非关键进程
- 影响实时系统的性能的两个关键因素
  - 中断延迟：CPU收到中断到中断处理程序开始的时间
    - 缩短中断禁用的时间
  - 调度延迟：调度程序从停止一个进程到启动另一个进程所需时间
    - 用抢占式内核降低调度延迟
- 抢占式优先级调度算法

算法评估

- 确定性建模：计算平均等待时间
- 排队模型
  - n为平均队列长度，W为队列平均等待时间，λ为新进程到达队列的平均速率

    可知 $n = \lambda * W$ （little公式）
- 模拟
- 实现

Chapter 5 线程

概述

- 线程拥有自己的ID，程序计数器，寄存器和堆栈
- 线程之间共享代码段、数据段和其他资源，如文件、信号
- 线程产生的动机：一个应用程序需要执行多个类似的任务
- 线程的优点：
  - 响应性：即便部分阻塞，也可以继续响应用户请求
  - 资源共享：线程默认共享地址空间、存储和其他资源
  - 经济：创建、切换线程开销较小
  - 可伸缩性：线程可以在多处理核上并行运行

并行类型

- 数据类型：将数据分布于多个核上，如分区域累加
- 任务并行：将任务（线程）分布到多个核上

多线程模型

- 线程支持的两种方法：用户线程和内核线程
- 多对一模型：多个用户线程映射到同一个内核线程（容易阻塞），如GNU thread线程库
- 一对一模型：每个用户线程映射到一个内核线程（减少阻塞），如linux
- 多对多模型：多路复用多个用户级线程到同样数量或更少数量的内核线程，如Windows NT/2000
  - 双层模型：可以多对多，可以一对一

多线程问题

- 隐式多线程：将多线程的创建于管理交给编译器和运行库来完成
- fork和exec：如果调用exec，那么这个程序将取代整个进程，包括所有线程
- 线程撤销：
    - 异步撤销：一个线程立即终止目标线程
    - 延迟撤销：目标线程不断检查它是否应该终止
- 信号处理
    - 信号的原则
        - 信号是由特定事件的发生而产生的
        - 信号被传递给某个进程
        - 信号一旦收到就应该处理
    - 信号处理程序
        - 缺省的信号处理程序
        - 用户定义的信号处理程序
    - 线程信号处理的方式
        - 传递信号到信号所适用的线程
        - 传递信号到进程内每个线程
        - 传递信号到进程内某些线程
        - 规定一个特定线程接收所有信号
- 线程池：将创建好的线程放到一个池中等待任务
    - 优点
        - 用现有线程服务请求比创建一个线程更快
        - 线程池限制了任何时候可用线程的数量
        - 将要执行任务从创建任务的机制中分离出来，允许采用不同策略运行任务

线程调度

- GCD大中央调度：允许开发人员将某些代码区段并行运行
    - 将块放在调度队列上，有串行队列和并行队列

Chapter 6 进程同步

临界区问题

- 临界区：进程执行这段代码时可能修改公共变量、更新一个表或者写一个文件
- 临界区问题的解决方案要求：
    - **互斥**：同一时间只能有一个进程在临界区执行
    - **进步**：如果没有进程在临界区内执行并且有进程需要进入临界区，那么必须选择一个进程进入，且选择时间不能无限长
    - **有限等待**：从一个进程请求进入临界区到这个请求被允许，其他进程允许进入临界区的次数有上限

Peterson解决方案

- 要求共享两个数据项：

```
int turn;//共两个进程
bool flag[2];
do{
    flag[i] = true;
    turn = j; // j = 1 - i
    while(flag[j] == true && turn = j);
    //临界区
    flag[i] = false;

}while(1);
```

硬件同步

- 单处理环境：关中断
- 多处理环境：当前计算机提供特定的原子硬件指令，如交换指令

信号量

- 整形信号量：
  - 初始化为N，N为可使用的资源数
  - 每当有wait时，信号量递减，直到等于0时，wait开始自旋或挂起
  - 有signal时，信号量递增
- 二进制信号量：
  - 只有0和1，可以用来作互斥锁
- 以上两者的缺点：忙等（但多线程中可以引入忙等）
- 记录型信号量
  - 大于0：表示资源个数；等于0：临界；小于0：绝对值表示等待队列长度
  - 递减后信号量小于0时，需要在wait中借用block调用来阻塞，并将其加到等待队列

    递增后信号量小于等于0时，需要在signal中借用wakeup调用来唤醒，并将其移出等待队列

生产者消费者问题

- 初始时empty = N，full = 0
- 生产者：等（减）一个empty，放（加）一个full，互斥锁要紧贴

```
while (true) {
    // produce an item
    wait (empty);
    wait (mutex);
    // add the item to the buffer
    signal (mutex);
    signal (full);
}
```

- 消费者：与生产者相反

```
while (true) {
    wait (full);
    wait (mutex);
    // remove an item from buffer
    signal (mutex);
    signal (empty);
    // consume the removed item
}
```

读者写者问题

- 写者：用写自旋锁保护自己写操作打断

```
while (true) {
    wait(wrt_mutex);
    // writing is performed
    signal(wrt_mutex);
}
```

- 读者：用自旋锁保护readcount修改不被干扰，同时保证等待读的只有一个读者

```
while (true) {
    wait(mutex);
    readcount ++;
    if (readcount == 1)
        wait(wrt_mutex);
    signal(mutex)
    // reading is performed
    wait(mutex);
    readcount --;
    if (readcount == 0)
        signal(wrt_mutex);
    signal (mutex);
}
```

哲学家就餐问题

管程

- 管程确保了每一次只有一个进程在管城内处于活动状态

Chapter 7 死锁

死锁特征

- 死锁的必要条件
    - 互斥：至少有一个资源处于非共享模式，即只有一个进程可以使用，其他进程必须等待
    - 占有并等待：一个进程应占有智少一个资源，并等待另一个资源
    - 非抢占：资源不能被抢占
    - 循环等待：n个进程循环等待下一个进程的资源
- 资源分配图 书P215

- 如果分配图没有环，那么系统就没有进程死锁

  如果分配图有环：

  - 如果每个资源类型恰好有一个实例，那么就出现了死锁

死锁预防

- 互斥：可共享资源不会参与死锁
- 持有且等待：要求进程申请资源的系统调用在所有其他系统调用之前进行，来使得每个进程在执行前申请并获得所有资源
- 非抢占：若一个进程持有资源并申请另一个不能立刻分配的资源，那么它现在拥有的资源都可以被抢占
- 循环等待：对所有资源类型进行完全排序，要求每一个进程按照递增顺序来申请资源

银行家算法

书P223

Chapter 8 内存管理

内存与程序

- 地址绑定
  - 编译时：绝对代码
  - 加载时：可重定位代码
  - 执行时：内存段转移
- 逻辑地址空间与物理地址空间
  - 内存管理单元（重定位寄存器）

交换

- 将进程暂时地从内存交换到备份存储，也可以反向交换，可能增加多道程序度
- 交换可能让总的物理地址空间大于内存的物理地址空间

连续内存分配

- 动态存储分配问题
  - 首次适应：分配首个足够大的孔
  - 最优适应：分配最小的足够大的孔
  - 最差适应：分配最大的的剩余孔
  - 循环首次适应：分配下一个足够大的孔
- 碎片
  - 外部碎片：当总的可用内存之和可以满足请求但不连续时，就出现了外部碎片问题：存储被分成大量的小孔
    - 紧缩：移动内存内容进行合并
    - 允许进程逻辑地址空间是不连续的（分段分页）
  - 内部碎片：进程分配的内存可能比所需要的大

分页

- 逻辑地址的分离：高位为页码，低位为页偏移
- 页表：每个进程都有一个页表副本，用页码作索引，转换到正确的物理地址进行访问
- 分页没有外部碎片
- TLB只记录最后一层页表，没有分层TLB
- 分层页表中，使用中最高几位作一级页表索引，查出二级页表的页码。接下来几位是二级页表的页偏移
- 有效访问时间公式：TLB命中率 * 主存访问时间 + TLB失效率 * （页表访问时间+主存访问时间）
- 保护：页表中的有效位和无效位

分段

- 段表中保存了界限和基地址，将物理内存划分为了不同长度的段
- 地址被分为两部分：段号和段偏移
- 如果对段有非法引用，则会陷入操作系统

Chapter 9 虚拟内存管理

虚拟存储器的特征

- 多次性（最重要）：一个作业被分成多次装入内存运行
- 对换性：允许在进程运行过程中换入换出
- 虚拟性：逻辑上的扩充

按需调页

- 缺页错误：对标记为无效的页面进行访问会产生缺页错误
- 缺页错误的处理方式：
    - 检查进程内部表，确定该引用有效或无效
    - 若无效，终止进程；若有效，则调入页面
    - 找到一个空闲帧
    - 调度一个磁盘操作，将所需页面读到刚分配的帧
    - 当磁盘读取完成时，修改内部表和页表
    - 重启被中断的指令
- 性能：缺页处理时间主要有三个组成部分
    - 处理缺页错误中断
    - 读入页面
    - 重新启动进程
    - 性能的提升方法
        - 减少缺页错误处理时间：使用交换空间、利用脏位表、开始时从文件系统调页，之后写入交换空间
        - 降低缺页错误率：增大页面、增强程序局部性

写时复制

- 父进程和子进程共享页面，只有当任何一个进程写入共享页面，才会创建这个页面的副本

页面置换

- 基本页面置换
  - 找到所需页面的磁盘位置
  - 找到一个空闲帧或选择一个牺牲帧
  - 将所需页面读入空闲帧，修改页表和帧表
  - 重启用户进程
- 选择牺牲帧时，可以借助脏位来减少输入输出时间
- 算法：
  - FIFO页面置换：容易引发Belady异常，帧数多反而失效率高
  - 最优页面置换：找到未来最长时间不会使用的页面
  - LRU页面置换：最近最少使用的页面，无Belady异常，称为堆栈算法
  - 额外引用位算法：循环右移，找到最小的编号就是需要置换的页面
  - 第二次机会算法：检查引用位，如果是1，获得第二次机会并清除引用位；如果是0，直接替换
  - 增强型第二次机会算法：将引用位和修改为合起来考虑
  - 计数页面置换
    - 最不经常使用
    - 最经常使用
- 页面缓冲：写出牺牲帧前，所需页面就已经读到来自缓冲池的空闲帧里，可以尽快启动

帧分配

- 帧的最小数：必须要有足够的帧来容纳任何单个指令可引用的所有不同页面
- 帧的最大数：可用物理内存数目
- 分配算法：
  - 平均分配
  - 比例分配：按照进程大小分配
- 页面置换算法的分类：
  - 全局置换：可以从别的进程获得帧——进程控制不了自己的缺页错误率
  - 局部置换：只能牺牲自己的帧

系统抖动

- 进程调页时间多于执行时间，那么它就在抖动
- 根本原因：不合理的多道程序度
- 限制方法：局部置换算法或优先级置换算法
- 工作集：监视最近k个页面，构成局部工作集，每个进程都有一个工作集，只要包含的页面足够少，就开启新的进程，否则将挂起一部分进程

分配内核内存

- 内核应保守的使用内存，最小化碎片浪费
- 伙伴系统：2的幂分配器，不断将长度除以2，找到最适合的分配长度
- slab分配器：把内核看做大小不一的对象，每一个slab都是若干连续物理页面，使用高速缓存来给内核对象分配空间

Chapter 10 文件系统

文件概念

- 文件是一系列字符记录
- 文件属性：名称、标识符、类型、位置、尺寸、保护、时间、日期和用户标识
- 文件操作：创建文件、写文件、读文件、重新定位文件、删除文件、截断文件
- 打开文件表：操作系统用来维护所有打开的文件信息
    - 每个进程表：每个条目指向系统打开文件表
    - 整个系统表
- 打开文件的关联信息：
    - 文件指针
    - 文件打开计数：每个进程打开一次，则加一
    - 文件的磁盘位置：查找磁盘上文件所需的信息保存在内存中
    - 访问权限：允许或拒绝后续I/O请求
- 文件锁：共享锁和独占锁
- 文件锁定机制：UNIX建议，Windows强制
- 文件类型：操作系统不一定支持
    - UNIX系统采用文件开始部分的幻数来表明文件类型
- 文件结构
    - 现代操作系统支持最小数量的文件结构
        - 应用可以更加灵活
        - 简化了操作系统
- 文件内部结构：将多个逻辑记录包装到物理块中
    - 会产生一定量内部碎片

访问方法

- 顺序方法：read/write next
- 直接访问（相对访问）：必须用块号作为参数，read/write next
- 索引访问方式：先访问索引表，根据对应项的地址来找到文件

磁盘与目录的结构

- 磁盘可以划为不同分区，每个分区由文件和目录组成
- 单级目录：所有文件都包含在同一目录下
    - 当文件数量增加或系统有多个用户时，单级目录有限制
- 两级目录：主文件目录（MFD）中记录了用户名，区分不同用户，下属的用户文件目录（UFD）需要使用路径名
- 树形目录：使用绝对路径名和相对路径名，删除策略不同

- 无环图目录：方便共享目录，采用“链接”的方法指向另一子目录
    - 共享文件的删除：
        - 直接删除：留下野指针
        - 搜索需要删除的链接：成本较高
        - 保留文件，直到所有引用全部删除
- 通用图目录：避免自我引用，需要垃圾收集方案

文件系统安装

- 操作系统需要知道设备的名称和安装点

文件共享

- 采用文件所有者和组的概念，所有者可以修改，组拥有相同访问权限
- 远程文件系统：
    - 使用网络来远程访问
    - 客户机-服务器模型允许一台计算机安装多台远程机器上的若干文件系统
    - 分布式信息系统对远程计算所需信息提供统一访问
- 故障模式：
    - 文件系统磁盘顾顺章、目录结构或其它磁盘管理信息（统称为元数据）损坏和其他损坏
- 一致性语义：用于评估支持文件共享的文件系统，规定系统的多个用户如何访问共享文件

保护

- 可靠性：通过文件的重复副本来提供，自动定期将可能意外损坏的文件系统复制到磁带
- 安全性：拒绝不合适的访问
    - ACL：访问控制列表
    - 三种用户类型：所有者、组、其他

Chapter 11 文件系统实现

文件系统结构

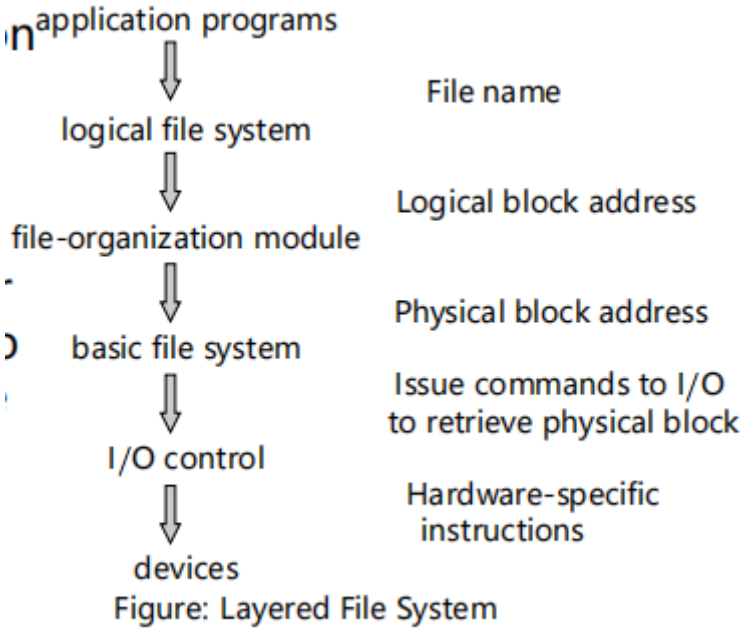
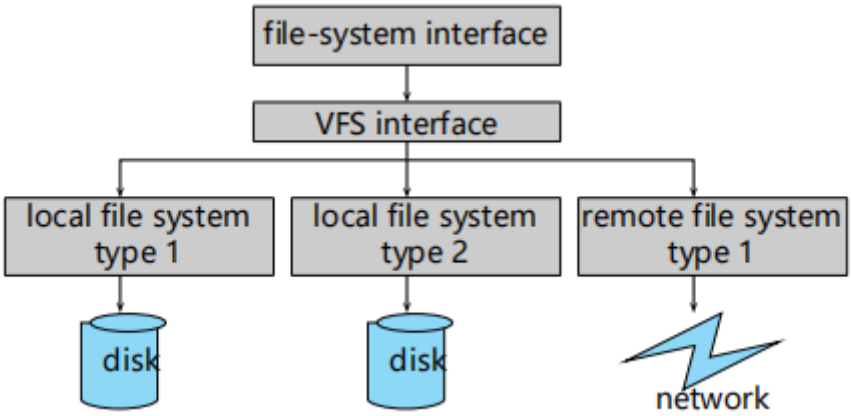


Figure: Layered File System

- I/O控制：包括设备驱动程序和中断处理程序，在主内存和磁盘系统之间传输消息
- 基本文件系统：向设备驱动程序发送命令，以读取写入物理块
- 文件组织模块：将文件逻辑块地址转换为物理块地址
- 逻辑文件系统：管理元数据信息，包括文件系统的所有结构而不包括实际数据

文件系统实现



- 第一层：文件系统接口，基于文件系统调用
- 第二层：虚拟文件系统VFS，将通用操作和实现分开，基于虚拟节点来唯一表示一个文件
    - 区分本地文件和远程文件，根据文件系统类型，进一步区分本地文件

目录实现

- 线性列表：采用文件名称和数据块指针的线性列表
- 哈希表：根据文件名称获得一个值，返回线性表内一个元素指针

分配方法

- 连续分配：每一个文件都占据了磁盘上连续的块
  - 优点：支持随机和连续访问
  - 缺点：外部碎片，文件大小不能扩展
- 链接分配：
  - 文件分配表：每个磁盘块都有一个条目，可以用块号索引

    FAT需要高速缓存
- 索引分配
  - 将所有指针放在一起形成索引块
  - 优点：随机访问，动态访问无外部碎片
  - 缺点：索引块指针开销，文件大小限制

存储空间管理

- 空闲表法
- 空闲链表法
- 位示图法
- 成组链接法