第一章

操作系统是什么?

- 控制和管理整个计算机系统硬件和软件资源
- 合理组织调度计算机资源分配
- 提供给用户较为方便的接口和环境

操作系统的特征

- 并发：两个或者多个事件在同一时间间隔内发生
- 共享：系统中的资源可供内存中多个并发进程使用
- 虚拟：将一个物理上的实体通过逻辑设计拆成不同的对应物
- 异步：每个进程运行的速度无法预知

设计目标

- 有如下功能
    - 管理处理器（CPU）
    - 管理存储器（磁盘）
    - 管理设备（IO）
    - 管理文件
- 作为用户与计算机之间的接口
    - 使用命令接口控制作业
    - 使用程序接口（系统调用命令）控制作业
        - 相当于用一个接口去执行一些机器指令实现的功能

发展阶段

- 人工操作阶段：人力来调整电路
- 单道批处理阶段：按顺序处理一批作业
- 多道批处理阶段：允许并行处理一批作业
    - 分时系统进行调度（用户以时间片为单位轮流使用CPU）

作业与进程

- 进程就是某个在跑的程序
- 作业就是几个程序一起完成一个任务（比如用批处理方式连续执行几个程序）

微内核

- 宏内核就是把各种各样的东西全部包裹进去
- 微内核从用户那里收到消息，然后将消息发送给指定的模块。这样只需要保留最小的一个OS系统，其他的让其他模块去做。

第二章

为什么要引入进程?

- 在多道程序环境下，允许有多个程序并发执行，此时程序失去封闭性，并具有间断性与不可再现性。因此引入进程来描述如何控制程序并发执行，实现操作系统的并发和共享

进程的组成:

- PCB：描述进程基本状况
- 程序段：进程要执行的操作
- 数据段：栈、全局变量等

进程的特征:

- 并发性
- 动态性：动态产生动态消亡，有一定的生命周期
- 独立性：作为资源分配的独立单元
- 异步性
- 结构性：用PCB进行描述

为什么要引入线程?

- 创建一个进程太太太麻烦了：要申请开内存、开各种表维护代码、堆栈。同时在轮转中断时要保存太多的状态。进程要通信必须开管道or用message，而线程可以直接共享同一进程下的数据空间。
- 线程就是“轻量级进程”，作为进程中的一个实体（程序执行流的最小单元），被系统独立调度和分派。进程只作为除CPU以外的系统资源分配单元；线程则作为处理器的分配单元
- 多个线程共享一个进程。线程就只有程序计数器，一些寄存器和栈（最基本的让程序跑起来的玩意），

线程与进程的比较:

- 调度：在引入线程的操作系统中，线程是独立调度的基本单位，进程是资源拥有的基本单位。
- 并发性：进程和线程之间都可以并发执行。
- 系统开销：线程比进程开销小的多（开销极小）

进程和程序比较:

- 进程能更真实地描述并发
- 程序是静态的，进程是动态的；进程有生命周期，程序相对来说不删掉就不消亡
- 一个程序可以对应多个进程
- 一个进程也可以产生多个进程
- 进程创造进程（fork），程序不创造进程

进程五态:

- 创建状态（new）：进程首先申请一个空白的PCB，然后在PCB填入控制与管理进程的信息；系统接着为该进程分配相关资源，最后将进程转入就绪状态
- 就绪状态（ready）：进程已经处于准备运行状态，只要获得CPU就可以运行
- 阻塞状态（waiting）：进程因为缺乏某些数据（比如等IO）或者需要输出东西时就等待。即使处理器空闲也不能去运行
- 运行状态：进程在处理器上运行，一个处理器同一时刻只能运行一个进程
- 结束状态：进程正常结束或者某些原因中断。此时系统先置该进程为结束状态，然后进一步释放资源。

PCB：记录进程的一些信息（如状态，计数器，使用的寄存器，分配的CPU使用优先级，内存使用信息，IO状态信息，文件信息）：

- 进程描述信息：如进程标识符、用户号，父子进程关系
- 进程控制信息：如进程处于五态的哪个状态，使用CPU优先级
- 进程资源信息：使用了哪些文件、寄存器、IO设别

OS把所有PCB组织在一起，就构成了PCB表。PCB表的大小除以每个PCB大小记录系统中最多有多少个可同时存在的进程。（称为并发度）

进程上下文（context）：在进程要被中断切换时记录当前进程运行的状态

进程调度：

- 高级调度：调度作业（切批处理）
- 中级调度：调度进程（把进程切换入内存或外存），为当前进程执行申请内存，将缺少的数据换入内存（切主辅存）
- 低级调度：（毫秒级的调度）处理机选择就绪的进程或线程进入运行状态（切CPU）

进程控制原语：

- 创建原语：
    1. 为新进程分配一个进程标识号，申请空白PCB
    2. 为新进程分配存程序和数据的资源
    3. 初始化新进程的PCB
    4. 一切准备好后将新进程插入就绪队列
- 终止进程（撤销原语）：当进程正常结束或者触发某种异常事件（存储越界，缺少permission，IO出错等）
    1. 根据被终止进程标识找到它的PCB，读出该进程状态
    2. 如果是执行状态立刻中断进程，将资源分配给其他进程
    3. 递归终止其他子进程
    4. 将资源归还给父进程或者操作系统
    5. 删除PCB
- 阻塞原语：
    1. 找到要被阻塞的进程的PCB
    2. 若运行，则中断并记录当前状态

3. 从当前队列移到阻塞队列

- 唤醒原语：
  1. 从等待队列找到相应PCB
  2. 将它移到就绪队列，让算法想办法调度

进程的联系：

- 竞争：两者之间没有关系，相互无感知，竞争CPU资源
- 共享：两者之间共享第三方资源，通过共享资源进行合作
- 通信：两者相互交流，通过通信协作

由此得出同步概念：同步即进程之间有相互合作关系（某一个人要干活必须等另一个人输出结果），他们的工作次序必须是固定的，因而相互制约

互斥：两个进程要访问同一个资源，一个人只能访问一个资源，访问操作互斥

进程的通信：

- 共享存储：在共享内存里通过某个数据结构或者存储区来传递数据
- 消息传递：通过格式化的message进行数据交换
  - 直接通信：send原语直接指定接受进程的pid，然后中断自己。操作系统将message传到一个缓冲区里，然后将该缓冲区挂上receive进程的消息链。最后发送进程继续执行
  - 间接通信：（两个进程要share同一个mailbox）send原语将message发送给mailbox（每个mailbox都有一个id），receive从mailbox里获取message。
    - mailbox：由操作系统管理，可以单向（必须A送B收）也可以双向。信箱记录可存放信件数，已存放信件数，指针（迭代）。信箱满则send进程等着信箱空；信息空则receive一直等着信件。
- 管道通信：利用一个共享文件来连接两个进程。一个进程write文件，另一个进程read。

进程调度：

- 不能调度：中断处理、进程在临界区（加了锁，应该尽快执行完毕）、原子操作
- 可以调度：引发调度条件or运行不下去时调度（非剥夺），中断处理or自陷处理结束后如果说了想调度可以立刻进行调度（剥夺式）
- 调度方式
  - 非剥夺：某个进程正在CPU上跑，即使另一个进程想进来也不让进
  - 剥夺：可以立刻打断当前进程换上另一个进程（需要符合优先级）
- 调度优先级准则：
  - CPU利用率：尽量让CPU一直在跑
  - 系统吞吐量：短作业可以快速完成吐出结果，因此让短作业先跑可以让任务吞吐量加快
  - 周转时间：即作业提交到作业完成，包括等待，就绪队列等，处理器上跑和输入输出。尽量让所有任务周转时间在一个合理的水平。
  - 等待时间：指进程等处理器时间。应该让每个进程等待时间都不太久
  - 响应时间：指用户提交作业到用户获得结果时间。

- 调度算法：
  - FIFS（First in First serve):有利于长作业不利于短作业，效率不高
  - SJF（short job first):优先处理短作业，长作业容易饥饿
  - 高响应比优先调度算法：响应比指（1+等待时间/服务时间），每次取出响应比最高的作业。这样可以让短作业（服务时间少）优先且长作业不至于饥饿。（等待时间越久调高权重）
  - 时间片轮转调度算法：每个进程只能运行一个固定的时间片。运行完就放到队尾
  - 多级反馈队列：先给一个任务2ms，跑完就结束，跑不完就丢给下一级队列（下一级队列一定要上一级队列跑完才能跑），下一级队列给你跑8ms，再跑不完再丢下一级。

线程组成：

- 基本不拥有系统资源
- 只保存程序计数器、寄存器和一组栈
- 不运行时保存上下文
- TCB：线程控制块，记录线程状态

用户级线程和内核级线程：

- 线程库：一些管理线程的API，有用户级的线程库和内核级线程库。
- 用户级线程：由用户搞的应用程序自己管理所有线程。内核只管这个进程，进程内部的线程由进程自己管理。因此线程切换不需要核心态特权，调度也由应用自己想办法。但是由于OS管理这个进程，所以进程时间片由所有线程共享；某个线程调用系统调用且系统调用阻塞时整个进程阻塞。
- 内核级线程：OS知道这些线程的存在，且由OS自己想办法控制这些线程。线程需要切换时就由用户态转为内核态，切换完由内核态返回用户态。没有线程库，但可以用内核提供的线程API进行创建等
  - 好处是由多个处理器时一个进程可以多个线程一起跑。一个线程阻塞了其他线程不受影响
  - 坏处是切换需要调用内核，增加开销。

多线程问题

竞争条件：多个进程并发访问同一数据且需要按顺序读写

临界资源：系统中某些资源一次只允许一个进程使用，这些资源叫做临界资源

- 临界区：需要访问临界资源的程序段叫临界区

实现临界区互斥的一些方法：

- 软件：
  - 单标志：首先让Pi进，其他等着，Pi进完Pj进，然后以此类推
  - 双标志：先检查另一个进程在不在临界区，不在就修改本进程临界区标志声明占用；退出的时候再改回来（两个线程都看到对方是false再进入，会冲突的错误算法）
  - 双标志plus：先修改自己的flag再检查别人在不在使用临界区（两个flag都true时会死锁的错误算法）
  - 标志+谦让：先声明自己要进，然后将使用者声明为对方。若自己为True且使用者标志为自己可以进，否则不能进。
- 硬件：

    - 用原子性语句上锁
    - 可以是lock语句，使用后释放lock
    - 也可以swap语句，使用完将使用权交给下一个。
    - 优点：解决方法简单，且支持多个临界区同时互斥
    - 缺点：上锁后用的进程就挂掉了会产生死锁；有可能有进程始终拿不到锁产生饥饿，每个进程都在等会出现一个进程跑八个进程都在耗CPU。

PV：P就是申请资源，V就是消耗资源

- 生产者消费者：full初始为0，记录有几个资源；empty初始为空间N，记录还剩多少空间。
    - 生产者：P(empty),P(mutex)(上锁),添加东西,V(mutex)(解锁),V(full)
    - 消费者：P(full),P(mutex)（上锁）,拿走东西,V(mutex)(解锁),V(empty)
    - 其实就是生产者每次操作消耗一个空位添加一个资源，消费者每次操作消耗一个资源添加一个空位。

死锁：所有的进程都等其他人释放资源

- **四个特征**：
    - **互斥**：每个资源只能被一个进程占有
    - **占有并等待**：每个进程都持有至少一份资源，且等待其他进程释放资源
    - **非抢占**：资源只能由占有他的进程释放，其他进程不能抢
    - **循环等待**：P0等P1，P1等P2，P2等P0
- 必要条件：申请边和分配边成环了。如果资源没有多份那就是充要条件。
- 解决方案：
    - 破坏占有并等待：要求程序申请资源时它本事不能占有任何资源
        1. 一次申请完所有要用的资源（由于所有要用的资源不是立刻准备好的，会产生饥饿以及运行过程时资源浪费）
        2. 可以申请一部分，但用完就立刻销毁。
    - 破坏非抢占条件：当某个进程申请资源申不到时，必须暂时释放已经拿到的所有资源，以后要的时候再申请。（开销很大，且有可能释放的资源又拿不到了，还得接着等）
    - 破坏循环等待条件：为每个资源编号，资源越少越紧缺的编号越大（意味着释放这个资源越紧急）。申请资源时必须从小到大申请（意味着当你需要更珍贵的资源时不那么珍贵的资源已经准备好了）
- 预防方案：在进程申请资源时系统判断它申请会不会死锁。
    - 判断是否为safe状态，意味着能否通过某种顺序分配资源满足每个进程对每种资源的最大需求。能够就可以放心分配
    - 如果找不到，就是unsafe，有可能死锁。可以用银行家算法检查。
- 检测死锁与恢复：如果出现了有环图，且没有办法在让某些节点释放资源后打断这个环，那么就说明死锁。
    - 恢复：剥夺一个进程占有的资源、回退进程取消申请、逐个撤销进程直到不死锁or把死锁的进程都kill了、重启系统。

        - 怎么选择干掉的进程?
            1. 进程优先级
            2. 剩余计算时间较长or总共计算时间短的
            3. 占有资源少的
        - 小心某个进程一直被你干掉进入starvation!

第三章

连续分配方式:

- 基本思想:把操作系统相关的东西保存在低内存区,用户进程保存在高内存区
- 方法:单一连续存储管理:
    - 用户用用户区,系统用系统区
    - 优点:好管理
    - 缺点:一次只能进一个任务,所有程序都装入,很少用的程序也要占内存
- 分区存储管理:
    - 基本原理:把内存分为几部分分区,os占一个分区,每个应用程序占几个分区。
    - 使用分区表(链表):记录空闲分区或者记录空闲和占用分区。一般就是空闲用一个链表记录每一块开始地址和长度、占用分区用链表记录开始地址和长度
    - 分区方法
        - 固定分区:大小相等就算每个分区的大小相同,大小不同就是多个小分区,适量中分区,几个大分区。
            - 比单一连续利用率高,允许多道程序同时进入内存
            - 缺点:分区内存在碎片,分区总数固定限制并发程序数目。
        - 动态分区:每次装入不固定大小的内存,而且运行过程可以继续申请内存。
            - 没有内部碎片,但外部存在碎片。
            - 首次适应算法:按地址递增找到第一个满足要求的空闲分区(低地址容易出碎片)
            - 最佳适应算法:按容量递增找到第一个满足要求的算法(最容易出碎片,且每次释放都要排序一次空地址)
            - 最坏适应算法:使用最大块的内存(容易消耗掉最大的连续内存)
            - 临近适应算法:首次适应,但每次从上一次申请内存的末尾开始遍历空区。
            - 释放算法:在将某个分区标记为空闲后要和相邻的分区合并成一个空闲分区。
- 内存扩充
    - 内存不够了辅存来凑。只常驻必要的代码和数据,其他部分平时放外存,要的时候调用
        - 没有调用关系的可以相互覆盖。
        - 或者交换暂时不能动的程序,把挂起or就绪的程序放到外存里。(可以交换进程,也可能是交换部分页)

- 页式管理
  - 把用户程序按页划分为大小相等的部分。页内从0开始编号，页号基于物理地址。通常高位是页号，低位是页内地址。
  - 内存块：页把内存划成一块块的。
  - 内存分配：以页为单位分配。注意逻辑相邻的页物理上不一定相邻（可能申请到一些碎片页）
  - 存储管理：进程页表：每个进程拥有一个页表，记录逻辑页号映射到哪一块内存块了；同时整个系统还有一个请求表，记录各个进程用了哪些页
    - 管理过程：先分析作业需要总块数N，然后检查有没有足够的内存块。如果有，那么给该进程的PCB填入一个页表地址和页表长度。然后页表记录用了哪些块和块的映射关系。最后修改使用了块的位图。
    - 硬件加速：快表（类似cache）
      - 往寄存器塞了页表。CPU给出页号和地址时直接去快表查内存块地址，找得到就直接获得指定地址，找不到就用替换算法把这个页号和地址加进去

虚拟存储：

- 原理：装入程序时只将一部分页读入内存就执行，执行过程中需执行的指令
- 好处：可以”虚拟“地扩充内存，从而能够逻辑载入更多的程序，进而并发
- 存储特征：虚拟地址使用不连续；块的调入调出是基于部分虚拟地址进行的；提供大范围的虚拟地址（容量大小为物理内存加外存交换区容量）
  - 多次性，对换性，虚拟性

实现：请求分页管理方式。

- 在程序内部有一个logical memory，比如程序“认为”自己有ABCDEFGH八块内存，现在要调入A，则首先查页表找到A所在的实际地址，然后将页A调入到内存里。
- 二级页表地址映射：页目录号在页目录地址中找到页表在的页目录，页表号在页目录里找到页表，最后根据页表地址计算偏移量找到指定的物理地址（就是二级页表）。

页面置换：

- FIFO：先进先出，选择最早建立的页面被置换
  - 性能差，容易抖动，较早调入的页往往是经常被访问的页
- OPT：最佳算法，选择未来不再使用的页调出
  - 理想化，每次未卜先知找到不用的页（其实没法实现，只能用来评价好坏）
- LRU：最近使用最少页面淘汰：选择最久未被使用的页面踢出去
  - 性能接近OPT，但硬件开销很大
- 最不常用算法：选择到当前时间为止被访问次数最少的页面
  - 每页都计数一段时间里被访问几次，如果产生缺页就踢掉最小的，然后归零重新计数。
- 轮转算法（最近未使用算法）：
  - 简单轮转：某页被访问就置use=1；指针转一圈找到use=0就踢出去，并将指针经过的use=1的页置0；如果都是1就转一圈踢掉最开头的

belady现象：如果某个进程没被分配它要求的所有页面，分配的越多反而缺页率越高

因此我们要优化调入分配清除策略

调入策略：

- 请求调页：发生缺页程序请求就给他调一页
  - 容易实现，但IO次数多
- 预调页：把这一页和相邻几页调入
  - 符合局部性原理
- 写回法进行页面调入调出，如果没有修改就直接覆盖下次从交换区里调，否则写回交换区。

清除策略：什么时候把表踢出去并写回外存

- 请求清除：某一页要被调出才写回
  - 调入还要调出写回，缺页等待时间增加
- 预清除：某页被置换前就根据某些算法优先被踢出去
  - 可能有不必要的开销（明明放在内存里一点事没有把别人踢出去）

缓冲池

存储形式：磁带，磁盘（把磁带盘起来），光盘

影响磁盘访问因素：寻道时间、延迟时间、数据传送

磁盘调度算法：

- FCFS：先来先服务
- SSTF：最短寻道时间优先，优先选择距离当前磁头最近的请求
- 扫描算法：磁头按一个方向移动，有访求就服务；如果没有就改变方向继续等访求
- C-SCAN：到最大值直接跳回0

RAID：一组可以并行工作的磁盘构成的磁盘阵列

第四章

文件管理：管理文件、目录、分区

文件系统设计目标：高效、快速、方便的信息存储和访问功能

- 文件：一组带标识的、在逻辑上有完整意义的信息的序列。
  - 标识：文件名
  - 信息项：文件内容基本单位（由写者和读者来解释每个信息项是什么意思）
- 文件系统：是操作系统中统一管理信息资源的一种软件，保证文件的存储、检索、更新。
- 对于操作系统：
  - 分配、管理文件的存储空间
  - 实现文件按名存取：根据文件名就能打开一个存储空间

    - 实现文件信息的共享、加密和权限设置。
- 对于用户：
    - 提供处理文件信息的API
    - 提供文件信息
    - 提供IO接口
- 文件属性：名字，类型，路径，文件大小，权限，修改时间等
- 文件操作：create，delete，write，read,seek file,rename,get attribute.

文件结构：

- 逻辑结构：无结构文件（流式文件）和有结构文件（记录式文件）
    - 流式文件：就是一串字符。灵活性好
    - 记录文件：文件由若干个记录组成，每个记录由一个键，可以按键进行查找（比如B树）
- 物理结构：连续结构（顺序结构），链接结构，索引结构，主要和存储设备有关
    - 连续结构：磁带
        - 优点：简单，顺序存取速度快
        - 缺点：创建时就确定了大小。如果要增删需要换一个大的块，要重新移动分配。
    - 链式结构：FAT
        - 文件信息放在若干个不连续的物理块上，物理块用链表连接
        - 优点：方便插入删除
        - 缺点：不适合随机读取（要从头读到尾），搜索某一块内存块需要更多次搜索
    - 索引结构：文件信息放在若干不连续的物理块里，每个文件都有一个索引表，记录每一块内存的地址。第i个块记录在第i个条目中（如记录2、10、8，就是文件有三块，第一块在地址为2的地方，第二块在地址为10的地方）
        - 优点：保持链式方便增删的优点，允许动态增长删除，也保证了随机读取
        - 缺点：索引表占位置、寻道次数增加
- Unix索引表：前十项是记录十个存放内存块的地址；第十一项指向另一个内存块，内存块能够存放内存块大小/地址长度个内存地址；十一项不够十二十三项可以添加二级、三级索引。

目录

目录项的信息：

- 基本信息：文件名、类型、组织
- 地址信息
- 访问权限信息
- 使用信息

文件目录：把所有FCB组织在一起就构成了文件目录

- 目录项：就是FCB，体现了文件的各种信息
- 目录文件：文件目录以文件的形式保存在外存，这个就叫目录文件

目录结构：为了能够更快的定位文件、分类文件、保证同名文件不被重复

一级目录结构：全部文件全存在一个目录下，没分组，没有考虑重命名

二级目录结构：在顶层目录下分了几个User目录

- 优点：搜索更快（每个用户只要在自己文件夹下搜）
- 缺点：没法grouping，没法逻辑分类

树形目录结构：多级目录

- 优点：层次清晰，可以分类，解决重命名
- 缺点：找一个文件需要逐层搜索；不利于文件共享

无环图：为文件增加了快捷方式

- 删除所有引用才删除文件or删一个引用就删文件

通用图：给目录增加快捷方式

- 存在环，可能无限搜索

文件访问办法：根据文件名搜索or根据路径名搜索

- 文件寻址：根据FCB上的文件地址去找到指定的内存块

目录优化：将目录中的FCB分为两部分，一部分记录文件名和文件号（次部），另一部分记录剩下的信息（除了文件名之外所有信息）。这样搜索的时候每个块能够装更多的信息，用文件符号名进行检索

空闲块表管理：

- 位图：用01记录这一块物理块是否能用
  - 注意，01只占一位。因此搜索方式为第几个字*字长+在字中的第几位。
- 成组链接法：把一些连续的块放在一起作为一组，每组之间再连接
  - 第一块记录有多少个块
  - 后面n-2个是后面块的地址
  - 最后一个记录下一块成组链接的地址

一些数据结构：

- 系统文件表：用于记录打开文件的信息
  - 记录FCB，文件号，是否被修改，共享计数
- 用户文件表：记录用户打开文件表的位置
  - 记录文件描述符，打开方式，读写指针，指向的系统文件表

文件操作：

- create：建立空的FCB然后记录相关内容

    - 检查参数合法
    - 检查是否重名
    - 检查有无位置
    - 记录FCB
    - 返回
- **open**：将FCB送到内存中
    - 首先根据给的路径找到FCB主部
    - 根据参数检查打开方式，共享说明和权限是否合法
    - 最后检查文件是否已经被打开
        - 没有打开就将这一个FCB写入系统文件表，并将共享计数记录为1
        - 如果已经打开就共享计数加一
    - 在用户文件表中记录打开方式等信息，并让其指向系统文件表的那一项
    - 返回读写指针（一个非负偏移量）
- Read：
    - 检查读取长度是否为正整数
    - 根据文件名找目录，然后找到文件
    - 然后判断权限
    - 然后检查要读的位置有没有超出指定位置
    - 最后根据读取长度找到指定的那一段所在的块号，读到内存缓冲区
        - 检查有没有跨块，如果跨块了就调入新的块

考前大题要点：

1. 概论

操作系统概念：

- 用户：操作系统是计算机用户使用的接口
- 虚拟机：操作系统是建立在计算机硬件上的虚拟机，提供更强的功能
- 资源管理：操作系统是管理计算机各类资源的管理者

因此设计目标：

- 用户觉得好用
- 系统管理高效

操作系统的特征：并发，共享，虚拟，异步

发展阶段：

- 人工操作
- 批处理

    - 单道批处理系统：可以**自动** 按**顺序**处理作业了
    - 多道批处理：宏观上可以并行执行作业了
- 分时操作系统：按时间片轮转的方式把处理器分配给各个作业（十几个用户共用一台主机，因此是人机交互的系统）
- 实时操作系统：实时性和可靠性（比如航天系统，一定要在指定时刻进行指定作业）

2. 体系结构

系统总线连着CPU，磁盘，打印机（IO），磁带（外存+IO），内存控制器

3. 作业

作业是一项任务，进程是作业实际执行的实体。一个作业可以由多个进程执行

4. 内核技术发展：单内核（所有功能放到一个内核）->分层内核（把内核分为多个层次，下级内核为上级内核提供功能）->微内核：提供一个最小的内核，其他的都给模块执行
5. 四种IO：
    - 同步阻塞IO：内核IO完成后才返回用户空间
    - 同步非阻塞IO：请求给IO后就可以立刻返回用户空间
    - IO多路复用（异步阻塞IO）：一个IO进程监听多个进程，有人准备好IO就通知系统进行读写
    - 异步IO：用户告知系统调用要一个IO，然后接着做；系统IO准备好了就中断程序开始IO
6. 进程和线程：
    1. 为什么要引入进程？多道批处理开始程序可以并行执行，此时程序失去封闭性，并有间断性（随时中断）与不可重复性（不能回推）。用进程可以更好地描述程序的并发。
    2. 为什么要引入线程？：由于创建一个进程需要的资源太多，要准备很多数据和内存空间，开销太大。因此引入线程。线程只包括了运行程序的最小部分（寄存器，栈），其他东西共享进程获得。
    3. 进程状态迁移：五态：创建、等待（阻塞）、就绪、执行、销毁
        - 等待是缺资源
        - 就绪是有资源但没被分配到CPU时间
        - 执行是有资源且被分配到CPU时间
    4. 进程的组成：程序段，数据段，PCB
        - PCB记录了进程的标识、进程在哪个状态、进程占用哪些资源
    5. 进程关系：同步（间接感知），互斥（没有感知），通信（直接感知）；临界区是访问临界资源的那段代码；互斥访问可以通过信号量或者条件变量（软件就是无锁算法，硬件就是原子性加锁）
    6. 高级通信：直接通信和mailbox，一个是挂载到receive进程的message队列，另一个是挂载到mailbox的队列
    7. 进程调度：FCFS，SJR，优先权调度（等待的进程优先级上升），轮转调度（公平分配时间片），多级队列调度（把不同任务放到不同队列），多级反馈队列调度（不同任务可以移动到不同队列里）

        - 引发调度：某个进程从运行切换到等待or就绪or终止；某个进程从等待切换到就绪（等CPU）
7. 动态分区分配：最先适配（循环最先适配）、最佳适配、最坏适配
8. 虚拟存储器：具有请求调入与置换功能，能够在逻辑上对内存容量进去扩充的存储器系统。逻辑容量由内存容量和
    - 多次性（多次装入），对换性（对换内存块），虚拟性（其实没那么大）
9. 请求分页的算法：先进先出，OPT，LRU，LFU（最不常用，least frequence use），轮转
10. 什么是文件？：有标识的，逻辑上有完整意义的序列
    - 文件名：标识
    - 信息项：基本单位
11. 什么是文件系统？管理组织文件，提供存储，更新和修改的API
12. 文件逻辑结构：流式文件和记录文件
13. 文件物理结构：连续（顺序）结构，链式结构，索引结构
14. 综合索引优点：可以顺序存取，也可以随机存取；优化插入删除；充分利用外存
15. 成组连接法：组内记录空闲块组间链表
16. 目录文件组成：基本信息，地址信息，权限信息，使用信息。目的：管理文件目录
17. 影响磁盘访问因素：寻道时间（找到磁道）、延迟时间（找到扇区）、传输时间（读写速度）