操作系统

选择题（10*2=20）

- 批处理系统提高吞吐率，优先选择耗时短的作业
- 批处理系统中，作业调度从后备作业队列中选出若干作业调入内存
- 批处理系统的主要缺点是缺乏交互性 +1
- 引入多道程序并发机制是为了提供CPU的使用效率
- 提高单机资源利用率的关键技术是多道程序设计技术
- 单处理机系统中，进程和进程不能并行，并行指同一时刻
- 实时操作系统必须在被控制对象规定时间内处理来自外部的事件
- 在分时系统中，时间片一定时，进程数越多，响应时间越长
- 采用时间片轮转调度算法，是为了多个终端和用户都能得到系统的及时响应
- 操作系统是一种系统软件 +1
- 哪一项不是操作系统的核心功能
- 关于操作系统描述不正确的是
- 用户程序要把字符送到显示器上，要使用操作系统提供的系统调用
- 原语不能被系统中断

- 哪一个不是进程的特征
- 进程状态信息存放在PCB中，PCB中存有PID，PCB存在于内核的内存空间中通过进程树联系
- 进程状态转移 +1 +1 +1 +1 +1 +1
- 进程和程序的本质区别 动静态 +1 +1 +1 +1
- 进程与线程的关系，用户级线程与内核级线程的关系
- 临界区概念 空闲让进，互斥使用，有限等待
- 用信箱实现进程间通讯的通讯原语是发送原语和接收原语
- 死锁预防的方法 资源有序分配法破坏循环等待条件
- 资源有限时，发生死锁的必要条件
- 分析那种情况不会产生死锁
- 银行家算法是一种死锁避免方法
- 资源有序分配法破坏死锁的循环等待条件 +1
- PV管理临界区，信号量初值一般为一 +1
- 信号量变化范围 +1 +1

- 动态分区内存管理中，首次适应算法倾向于优先使用低地址部分空闲区
- 页面置换算法FIFO，Belady
- 动态重定位的时机（执行） +1
- 缺页中断处理后应执行被中断的那一条指令 +1
- 实现虚拟内存的目的，扩充逻辑内存
- 页式存储采用二级页表并不使用快表，则要访存几次 +1
- windows消息调度机制是消息队列
- 缓冲技术用于提高主机与设备交换信息的速度，缓和速度不匹配的矛盾，解放cpu
- 段式存储管理长度
- 作业等待时间不会对缺页中断次数产生影响
- 设计多级页表结构的目的是减少页表所占用的连续内存空间
- 混合索引结构管理磁盘空间大小

- 工作集的判断

- 将文件设置成所有人可读取的命令是chmod ugo+r file1.txt
- 节省存储空间不属于文件系统采用多级目录结构的目的
- 文件分配方式，磁盘块的索引分配方式
- 无环图结构目录图示
- 用户删除文件时不可能删除文件所在目录
- 流式结构不属于物理结构的范畴

- 位示图的大小
- 位示图的作用 磁盘空间管理 +1 +1
- 才用SPOOLing技术的目的是将独占设备转成共享设备，提高设备的利用率
- 磁盘寻道算法访问次序
- 用户程序发出IO请求后，系统的处理流程（IO设备分层）
- 为改善磁盘IO性能不能将磁盘设置成多个分区
- IO子系统中环形缓冲可以有效处理IO burst的场景
- 通道是一种IO专用处理机
- 设备独立性的实现是通过逻辑设备名与物理设备名的映射，使其与物理设备无关
- 下列哪一个linux命令是修改文件和目录的权限、查看目录下的文件，第一次上机

简答题（5*4=20）

- 通道的作用和意义

> **通道的作用：** 专门负责I/O操作的处理器，解放CPU处理数据传输任务。
>
> **主要功能：**
>
> - 执行I/O指令序列
> - 控制外设与内存间的数据传输
> - 处理I/O中断
> - 实现多个I/O操作并发
>
> **重要意义：**
>
> 1. **提高CPU利用率**：CPU启动I/O后继续执行其他任务
> 2. **并行处理**：CPU计算与I/O传输同时进行
> 3. **批量传输**：通道可独立完成大量数据传输
> 4. **设备管理**：一个通道管理多个设备
>
> 通道是现代计算机实现高效I/O的关键技术，使CPU-I/O并行成为可能。

- 简述操作系统为什么要引入双模式，好处是什么？怎么进行双模式的切换（系统调用）

> **引入双模式的原因：** 区分特权操作和普通操作，防止用户程序直接操作硬件或执行危险指令。
>
> **主要好处：**
>
> 1. **保护系统**：防止用户程序破坏操作系统
> 2. **资源管理**：内核独占关键资源控制权
> 3. **故障隔离**：用户程序崩溃不影响系统
> 4. **安全控制**：实现权限管理和访问控制

> **模式切换过程（系统调用）：**
>
> 1. 用户程序执行陷入指令（如INT、TRAP）
> 2. CPU自动：
>   - 切换到内核态（修改PSW模式位）
>   - 保存用户程序状态
>   - 跳转到内核中断处理程序
> 3. 内核执行系统调用服务
> 4. 返回时恢复用户态和程序状态
>
> **触发切换的事件：**
>
> - 系统调用（主动）
> - 中断（硬件信号）
> - 异常（除零、缺页等）
>
> 硬件支持是关键：CPU通过模式位控制指令权限，特权指令只能在内核态执行。

- 说明**系统调用与库函数**的联系与区别 +1

> ### 1. 联系
>
> - **共同目标**：为用户程序提供功能支持（如文件操作、进程控制等）。
> - **封装关系**：
>   - **库函数可能封装系统调用**（如`fopen()`内部调用`open()`）。
>   - 并非所有库函数依赖系统调用（如纯计算的`sqrt()`）。
>   - **系统调用定义** 系统调用是应用程序请求操作系统内核服务的编程接口，是用户态程序访问内核功能的唯一途径。
>
> ### 2. 区别
>
> | 特性 | 系统调用（System Call） | 库函数（Library Function） |
> | --- | --- | --- |
> | **执行权限** | 内核态（通过软中断/陷阱指令触发，如`int 0x80`） | 用户态 |
> | **性能开销** | 高（需上下文切换） | 低（无模式切换） |
> | **功能范围** | 直接操作硬件/内核资源（如`read()`、`fork()`） | 可能仅处理用户态逻辑（如`strcpy()`） |
> | **可移植性** | 依赖操作系统（不同OS接口不同） | 跨平台（如C标准库`libc`） |
> | **实现位置** | 操作系统内核提供 | 用户态库（如`glibc`、`WinAPI`） |

- *简述多处理器调度（均衡、亲和等）

> 多处理器调度主要解决两个问题：
>
> **负载均衡**
>
> - 工作窃取：空闲CPU从忙碌CPU窃取任务

- 周期性重平衡：定期重新分配任务

CPU亲和性

- 软亲和性：优先在上次运行的CPU调度，利用缓存局部性
- 硬亲和性：强制绑定到特定CPU
- NUMA感知：优先本地内存节点调度

实现机制

- 每CPU本地队列 + 全局负载均衡
- 推送/拉取迁移策略
- 考虑缓存热度和迁移成本

目标是最大化吞吐量同时保持响应性和缓存效率。

- **操作系统**有哪些主要功能？操作系统结构有哪些？+1

1. **进程管理**
   - **功能**：进程创建、调度、同步、通信及死锁处理。
2. **内存管理**
   - **功能**：内存分配与回收、地址转换（如页表）、虚拟内存实现。
3. **文件系统管理**
   - **功能**：文件存储、目录组织、权限控制（如inode、ACL）。
4. **设备管理**
   - **功能**：设备驱动、I/O调度（如磁盘SCAN算法）、缓冲区管理。
5. **用户接口**
   - **功能**：提供命令行（Shell）或图形界面（GUI），系统调用封装。

---

1. 分层法
2. 模块化
3. 宏内核
4. 微内核
5. 外核

- 请简要说明进程创建的fork()和exec()系统调用的区别及作用。

fork()系统调用：

- 创建一个新进程（子进程），该子进程是父进程的完整副本
- 复制父进程的地址空间、代码段、数据段、堆栈等
- 返回值：父进程中返回子进程PID，子进程中返回0

exec()系统调用：

- 用新程序替换当前进程的地址空间
- 不创建新进程，而是将当前进程的代码和数据替换为新程序
- 原进程的PID保持不变

主要区别：

1. fork()创建新进程，exec()替换进程内容
2. fork()后有两个进程同时运行，exec()后仍是一个进程

> 3. fork()保留原程序继续执行，exec()丢弃原程序

- *进程的调度层次

> 进程调度分为三个层次：
>
> **高级调度** - 决定作业后备队列到就绪队列，控制进程总数
>
> **中级调度** - 外存的就绪队列到内存的就绪队列，提高内存利用率和系统吞吐量
>
> **低级调度** - 决定CPU分配给哪个就绪进程
>
> 三层协同实现系统资源的有效调度。

- *进程的组成和描述

> 进程组成：
>
> - **程序代码**：可执行指令
> - **数据**：全局变量、堆、栈
> - **进程控制块（PCB）**：进程ID、状态、寄存器值、内存信息等
> - **系统资源**：内存、文件、I/O设备
>
> 进程描述通过PCB实现，操作系统用它管理进程的调度、状态转换和资源分配。

- 进程与程序的区别是什么?进程控制块(PCB)存储了哪些关键信息?这些信息如何帮助操作系统管理进程?

> **进程与程序的区别：**
>
> - **程序**：静态的指令集合，存储在磁盘上的可执行文件
> - **进程**：程序的运行实例，动态的执行实体，占用系统资源
>
> 一个程序可产生多个进程（如多开浏览器），进程有生命周期（创建→运行→终止）。
>
> **PCB主要内容：**
>
> 1. **进程标识**：PID、父进程ID、用户ID
> 2. **处理器状态**：程序计数器、寄存器、PSW
> 3. **进程调度信息**：状态（就绪/阻塞/运行）、优先级、调度队列指针
> 4. **内存管理**：页表、段表、内存限制
> 5. **I/O状态**：打开文件列表、I/O设备分配
> 6. **记账信息**：CPU时间、实际使用时间
>
> **管理作用：**
>
> - **进程切换**：保存/恢复CPU现场
> - **调度决策**：根据优先级和状态选择下一个运行进程
> - **资源分配**：跟踪进程占用的内存、文件、设备
> - **进程通信**：维护IPC相关信息
> - **异常处理**：通过父子关系处理孤儿进程
>
> PCB是OS管理进程的核心数据结构，使OS能够控制和调度多个并发进程。

- 用户级线程(User-LevelThread)和内核级线程(Kernel-LevelThread)的主要区别是什么?各自的优缺点是什么?

> **主要区别：**
>
> - **用户级线程**：由用户空间的线程库管理，内核不感知
> - **内核级线程**：由操作系统内核直接管理和调度

> **用户级线程：** 优点：
>
> - 线程切换快（无需陷入内核）
> - 创建销毁开销小
> - 可定制调度算法
>
> 缺点：
>
> - 一个线程阻塞，整个进程阻塞
> - 无法利用多核CPU
> - 系统调用会阻塞所有线程
>
> **内核级线程：** 优点：
>
> - 真正的并行执行（多核）
> - 一个线程阻塞不影响其他线程
> - 内核可统一调度
>
> 缺点：
>
> - 线程操作开销大（需要系统调用）
> - 占用更多内核资源
> - 线程切换慢
>
> **混合模型**（多对多）结合两者优点：多个用户线程映射到多个内核线程。

- 请分析下面这段程序，共创建多少个进程，level1-level4分别打印出几次?请给出分析过程

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(){
pid t pid1, pid2, pid3;
pid1 =0; pid2 = 0; pid3 = 0;
pid1 = fork();
if(pid1 == 0){
    pid2 = fork();
    pid3 = fork();
}
else{
    fpid3 = fork();
    if(pid3 == 0)
        pid2 = fork();
}
if((pid1 ==0)&&(pid2 == 0))
    printf("level1\n");
if(pid1 != 0)
    printf("level2\n");
if(pid2 != 0)
```

```
    printf("level3\n");
if(pid3 != 0)
    printf("level4\n");
return 0;
```

> pid2 = fork(); 结束后 pid3 = fork(); 会在P1和P2中都执行
>
> 关键点：父进程中fork不为零，子进程中fork为0

- 请分析下面两段程序，分别给出两个程序打印出多少个“_”?请给出分析过程。

程序一：

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
    int i;
    for(i=0;i<2;i++){
    fork();
    printf("-");}
    return 0;
}
```

程序二：

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
    int i;
    for(i=0;i<2;i++){
    fork();
    printf("-\n");}
    return 0;
}
```

> 关键在于无换行导致缓冲区未刷新，fork时子进程复制父进程缓冲区。于是程序一输出8个，程序二输出6个

- 什么是线程？描述**线程与进程**的区别

> **线程**：
>
> - CPU调度的基本单位，是进程内的一个执行流。
> - 共享同一进程的地址空间和资源（如文件、内存），独立拥有栈、寄存器、PC。
>
> **与进程的区别**：
>
> 1. **资源开销**：线程创建/切换开销小（共享资源）；进程开销大（独立资源）。

2. **独立性**：进程间隔离（地址空间独立）；线程间可直接通信（共享内存）。
3. **崩溃影响**：线程崩溃可能导致整个进程终止；进程崩溃不影响其他进程。
4. **并发性**：多线程可并行执行（同一进程内）；多进程需依赖IPC通信。

- 试从调度性、并发性、拥有资源及系统开销几个方面，对**线程与进程**进行比较 +1

1. **调度性**

- **进程**：分配资源的基本单位。
- **线程**：CPU调度的最小单位，同一进程的线程间切换更快（无需切换地址空间）。

2. **并发性**

- **进程**：进程与进程，进程与线程可并发
- **线程**：线程与线程可并发

3. **拥有资源**

- **进程**：分配资源的基本单位，拥有独立地址空间、文件描述符、信号处理等系统资源。
- **线程**：共享进程的资源（如内存、文件），仅独享栈、寄存器等线程私有数据。

4. **系统开销**

- **进程**：创建、销毁、切换开销大（需分配/回收内存、更新页表等）。
- **线程**：创建、切换开销小（共享地址空间，仅需保存/恢复少量寄存器）。

- 有两个**并发进程**P1,P2，他们的代码如下，共享变量x，请分析并列出所有可能地z和c的结果

进程异步性

- 操作系统**并发进程**之间有哪两种**制约关系**，请对每种关系进行简要陈述

1. **互斥（Mutual Exclusion）**
   - **定义**：多个进程不能同时访问某一**临界资源**（如打印机、共享变量等）。
   - **特点**：
     - 资源具有**排他性**，需通过同步机制（如信号量、锁）保证一次仅一个进程访问。
     - 例如：生产者-消费者问题中的缓冲区访问。
2. **同步（Synchronization）**
   - **定义**：进程间因**执行顺序依赖**而需协调（如A进程需等待B进程完成某操作）。
   - **特点**：
     - 通过**条件等待**机制（如信号量、管程、事件）实现顺序控制。
     - 例如：读者-写者问题中写者优先的等待。

关键得分点：互斥→独占资源；同步→顺序依赖。

- 简述**进程**的五种**状态**模型中除了新建和退出外，还有哪三种基本状态，并描述出三种状态转化的典型原因 +1

1. **就绪（Ready）**
   - **定义**：进程已获得除CPU外的所有必要资源，等待被调度执行。
2. **运行（Running）**
   - **定义**：进程正在CPU上执行指令。
3. **阻塞（Blocked/Waiting）**

    - **定义**：进程因等待某事件（如I/O完成、信号量释放）而暂停执行。
4. **典型转换原因**：
    - **就绪 → 运行**：进程被调度器选中（如基于优先级、轮转等算法）。
    - **运行 → 就绪**：时间片用完（分时系统）或更高优先级进程抢占CPU。
    - **运行 → 阻塞**：进程主动请求资源（如读取文件）或等待事件（如用户输入）。
    - **阻塞 → 就绪**：等待的事件已发生（如I/O完成），被操作系统重新唤醒。

- *简述操作系统传递参数的三种方法

> 操作系统传递参数的三种主要方法：
>
> **1. 寄存器传递**
>
> 将参数直接存放在CPU寄存器中传递给系统调用。适用于参数较少的情况，速度最快但受寄存器数量限制。
>
> **2. 内存块传递**
>
> 将参数存储在内存的指定区域，然后将该内存块的地址传递给操作系统。适用于参数较多或大小不定的情况。
>
> **3. 栈传递**
>
> 将参数压入程序栈中，操作系统通过栈指针访问参数。这是最常用的方法，支持任意数量的参数，但相对较慢。

- **进程间通讯**方式有哪些？

> 低级通讯方式：
>
> - PV操作
>
> 高级通讯方式：
>
> - 共享存储
> - 消息传递
> - 管道通讯
> - 信号

- 什么是**临界区**，下面给出的实现两个进程互斥的算法安全吗，为什么？ +1 +1

> 临界区是访问共享资源的代码段，同一时间只允许一个或者多个进程进入
>
> 空闲让进，互斥使用，有限等待
>
> 不安全

- 如果**fork()**被调用成功，简述下面程序的结果

> fork()在父进程返回子进程的PID>0，在子进程返回0，失败返回-1

- 简述**信号量**的定义、关键操作及其实现原理

> **定义**：
>
> - 一种**同步机制**，用于控制多个进程/线程对共享资源的访问。
> - 由**整型变量 + 等待队列**构成，支持原子操作。
>
> ---
>
> 1. **P（wait）操作**（申请资源）：
>   - 信号量 `S--`，若 `S < 0`，则阻塞当前进程，加入等待队列。

2. **V（signal）操作**（释放资源）：
    - 信号量 `S++`，若 `S ≤ 0`，则唤醒等待队列中的一个进程。

- **原子性保证**：通过**硬件指令**（如TSL、CAS）或**关中断**实现P/V操作的不可分割性。
- **阻塞/唤醒**：由操作系统内核管理等待队列（如PCB链表）。

- 什么是**管程**？霍尔管程的语义是怎么样的？请简述

> **管程（Monitor）定义**：
>
> - 一种**高级同步机制**，封装了共享变量及其操作（过程/方法），确保同一时间仅一个进程/线程可执行管程内的代码。
> - 组成：
>   - **局部变量**（共享数据）。
>   - **条件变量**（`condition`，用于阻塞/唤醒进程）。
>   - **入口队列**（保证互斥访问）。
>
> ---
>
> **霍尔管程（Hoare Monitor）语义**
>
> 1. **互斥访问**：
>   - 任一时刻仅一个进程活跃于管程内，其他进程阻塞在入口队列。
> 2. **条件变量操作**：
>   - `wait(c)`：
>   - 调用进程阻塞并释放管程控制权，加入条件变量`c`的队列。
>   - `signal(c)`：
>   - 唤醒`c`队列中的一个进程，**立即切换执行**（Hoare语义关键），当前进程暂停并进入**紧急队列**（urgent queue）。
> 3. **紧急队列优先级**：
>   - 被`signal`唤醒的进程优先执行，执行完毕后再恢复原`signal`发起者。
>
> **关键得分点**：封装共享操作→Hoare的`signal`立即切换→紧急队列保障优先级。

- 什么是**死锁**，产生的原因是什么？产生的必要条件是什么？ +1

> **死锁**是指多个进程（或线程）在执行过程中，因**竞争资源**或**彼此通信**而陷入的一种**互相等待**的状态，导致所有进程都无法继续执行。
>
> ---
>
> 1. **系统资源的竞争**
> 2. **进程推进顺序非法**
>   - 请求和释放顺序不当
>   - 信号量使用不当
>
> ---
>
> 1. **互斥条件（Mutual Exclusion）**
> 2. **占有并等待（Hold and Wait）**
> 3. **非抢占条件（No Preemption）**
> 4. **循环等待（Circular Wait）**

- 简述采用虚拟内存的原因以及作用

> **虚拟内存** 的采用主要是为了弥补物理内存（RAM）的不足，并提高系统的稳定性和运行效率。其主要作用包括：
>
> 1. **扩展可用内存**：当物理内存不足时，虚拟内存允许系统使用磁盘空间（通常是硬盘上的交换文件或页面文件）作为临时的内存补充，从而防止程序崩溃或运行失败。
> 2. **多任务处理**：虚拟内存使多个程序可以同时运行，而不必严格受限于实际的物理内存大小。操作系统可以根据需要调整内存分配，保证不同进程的运行。
> 3. **内存管理优化**：虚拟内存使用分页技术（Paging）或分段技术（Segmentation），可以更高效地管理内存，提高内存的使用率，减少碎片化问题。
> 4. **进程隔离**：每个进程都能拥有独立的虚拟地址空间，避免进程之间的直接干扰，提高系统的安全性和稳定性。
>
> 简单来说，虚拟内存相当于给计算机提供了一个“备用存储区”，让系统在物理内存不足时仍能正常运作。这在现代计算机系统中是非常重要的一项技术。

- 简述**段式**存储管理中逻辑地址到物理地址的转换过程

> 1. **逻辑地址结构**：
>   - 形式：`(段号S, 段内偏移量W)`。
> 2. **查段表（Segment Table）**：
>   - 通过**段号S**索引段表，获取该段的：
>   - **基址（Base）**：该段在内存的起始地址。
>   - **段长（Limit）**：该段的最大长度。
> 3. **越界检查**：
>   - 若 `W ≥ Limit`，触发**段越界异常**（Segmentation Fault）。
> 4. **计算物理地址**：
>   - 物理地址 = **Base + W**（基址 + 偏移量）。
> 5. **访问内存**：
>   - 根据物理地址读写数据。

- **段式**存储管理，判断以下逻辑地址是否为有效地址，如果有效就计算物理地址 +1

> 关键在于判断段内偏移量和length的比较

- 某一页式系统，其页表存在主存中，对主存一次存取要1.5ms，问实现一次页面访问需要多少时间？若系统中有快表，平均命中率85%，页表项在快表中访存时间忽略不计，此时存取时间是多少？ +1

> ### 1. 无快表（TLB）时的存取时间
>
> - **页表在主存中**：每次访问需先查页表（1次主存访问），再访问数据（1次主存访问）。
> - **总时间**：
> $T_{\text{无快表}} = 2 \times 1.5\,\text{ms} = 3\,\text{ms}$
>
> ### 2. 有快表（TLB）时的存取时间
>
> - **快表命中（85%概率）**：直接通过TLB获取物理地址，**仅需1次主存访问**（访问数据）。
> - **快表未命中（15%概率）**：需先查页表（1次主存），再访问数据（1次主存）。
> - **平均时间**：
> $T_{\text{有快表}} = 0.85 \times 1.5\,\text{ms} + 0.15 \times 3\,\text{ms} = 1.275\,\text{ms} + 0.45\,\text{ms} = 1.725\,\text{ms}$

> **关键公式**
>
> - 无快表：$T = 2 \times 主存访问时间$
> - 有快表：$T = 命中率 \times 单次访问时间 + (1 - 命中率) \times 2 \times 主存访问时间$
>
> 1. **无快表**：一次页面访问需 **3 ms**。
> 2. **有快表（85%命中率）**：平均存取时间降至 **1.725 ms**。

- 请描述**虚拟**机制下**页面异常**处理的基本流程

> 1. **触发条件**：
>   - CPU访问虚拟地址时，MMU发现页表项**无效**（不存在或权限不足）。
> 2. **硬件处理**：
>   - 保存现场（PC、寄存器等），触发**页面异常中断**，转入操作系统异常处理程序。
> 3. **软件处理（操作系统）**：
>   - **检查原因**：
>   - **非法访问**（如写只读页）→ 终止进程。
>   - **有效访问**（页在磁盘或未分配）→ 继续处理。
>   - **分配物理页**：
>   - 若页未分配（如首次访问），分配新物理页并清零。
>   - 若页在磁盘（交换区），发起**I/O请求**调入内存。
>   - **更新页表**：
>   - 填写页表项，标记为有效，设置权限位。
>   - **必要时置换**：
>   - 若内存不足，按置换算法（如LRU）淘汰脏页（写回磁盘）。
> 4. **恢复执行**：
>   - 重新执行触发异常的那一条指令，MMU可正常翻译地址。

- *RAID技术

> **RAID技术**（独立磁盘冗余阵列）通过将多个物理磁盘组合成逻辑单元，提升性能、容量或可靠性。主要分为以下级别：
>
> **常见RAID级别：**
>
> 1. **RAID 0**（条带化）
>   - **特点**：数据分块并行写入多个磁盘。
>   - **优点**：读写速度快。
>   - **缺点**：无冗余，任一磁盘故障导致数据丢失。
> 2. **RAID 1**（镜像）
>   - **特点**：数据完全复制到另一磁盘。
>   - **优点**：高可靠性（允许一块磁盘故障）。
>   - **缺点**：容量利用率仅50%。
> 3. **RAID 5**（带奇偶校验的条带化）
>   - **特点**：数据与校验信息分布式存储，需至少3块磁盘。
>   - **优点**：兼顾性能与冗余（允许一块磁盘故障）。
>   - **缺点**：写入性能较低（需计算校验位）。
> 4. **RAID 6**（双奇偶校验）
>   - **特点**：类似RAID 5，但有两组校验，需至少4块磁盘。
>   - **优点**：允许两块磁盘同时故障。

    - **缺点**：写入开销更大。
5. **RAID 10**（1+0，镜像+条带化）
    - **特点**：先镜像再条带化，需至少4块磁盘。
    - **优点**：高性能+高可靠性（允许每组镜像中一块磁盘故障）。
    - **缺点**：成本高（容量利用率50%）。

- 简单描述**颠簸**现象产生的原因是什么？解决颠簸问题有哪些方法？

> 原因是频繁的页面置换，导致大部分时间都用于IO操作，驻留集太小
>
> 解决办法：增加驻留集的大小，增大页面大小，改进页面调度算法

- 简述典型的**目录**结构。目前最广泛采用的目录结构的是其中的哪一种，它有什么优点

1. **典型目录结构**：
    - **单级目录**：所有文件位于同一目录，简单但易冲突。
    - **两级目录**：用户隔离（每个用户独立目录），仍缺乏组织性。
    - **树形目录**：层次化结构（根目录+子目录），支持灵活路径访问。
    - **无环图目录**：允许共享子目录/文件（通过链接），但需避免环路。
    - **通用图目录**：完全灵活共享，需复杂管理（如垃圾回收）。
2. **最广泛采用**：**树形目录**（如Linux/Windows）。
    - **优点**：
        - 结构清晰，逻辑分层。
        - 支持绝对/相对路径，易定位文件。
        - 权限管理方便（目录级继承）。

- 描述**文件系统的分层**组织结构

1. **I/O控制层（设备驱动层）**
    - **功能**：直接与硬件交互，处理设备控制命令（如磁盘读写、缓存管理）。
    - **关键点**：屏蔽物理细节（如磁道、扇区），提供统一的块设备接口。
2. **基本文件系统层（物理I/O层）**
    - **功能**：向对应的设备驱动程序发送通用命令，管理磁盘块的读写，处理缓冲区缓存（如预读、延迟写）。
    - **关键点**：优化磁盘访问效率，减少实际I/O操作次数。
3. **文件组织模块层（存储策略层）**

- **功能**：实现逻辑结构与物理结构的转换。
- **关键点**：管理空闲空间（位图/链表），处理文件块的分配与回收。

4. **逻辑文件系统层（文件管理层）**

- **功能**：维护文件元数据（如FCB、inode），处理目录结构、路径解析与权限控制。
- **关键点**：提供文件逻辑视图（如命名、属性管理），隔离物理存储细节。

- 请列举**文件系统**中**目录**的几种典型结构，并从文件命名，文件分组，效率等角度，阐述每种目录结构的优劣

1. **单级目录结构**

- **结构**：所有文件存放在一个目录下。
- **优点**：简单，实现容易。

- **缺点**：
  - **命名冲突**：不允许文件重名。
  - **无分组**：无法分类管理文件。
  - **效率低**：文件增多时查找速度下降。

2. **两级目录结构**

- **结构**：每个用户拥有独立的目录。
- **优点**：
  - **解决命名冲突**：不同用户可同名文件。
  - **简单分组**：按用户隔离文件。
- **缺点**：
  - **用户内无分组**：同一用户文件仍混杂。
  - **共享困难**：跨用户访问需额外机制。

3. **树形目录结构**（多级目录）

- **结构**：层次化目录树，支持子目录。
- **优点**：
  - **灵活分组**：通过路径分类文件（如`/docs/work`）。
  - **允许重名**：不同路径下可同名文件。
  - **高效查找**：路径定位缩小搜索范围。
- **缺点**：
  - **访问复杂度**：需完整路径名，共享需符号链接或硬链接。

4. **无环图目录结构**

- **结构**：允许目录/文件被多个目录引用（如硬链接或符号链接）。
- **优点**：
  - **支持共享**：文件可出现在不同路径。
  - **节省空间**：避免重复存储。
- **缺点**：
  - **管理复杂**：需处理循环引用（符号链接可避免）。
  - **删除问题**：需引用计数判断是否释放空间。

5. **通用图目录结构**

- **结构**：允许任意目录间链接，可能形成环。
- **优点**：共享灵活性最高。
- **缺点**：
  - **需环路检测**：避免无限遍历。
  - **维护开销大**：需垃圾回收机制。

- 简述通用**目录**结构实现文件和目录的共享方式

通用目录结构通常指**树形目录**（层次化结构），支持父子关系与路径访问（如`/home/user/file`）。

- **核心组件**：
  - **目录项（Dentry）**：记录文件名与inode的映射。
  - **索引节点（inode）**：存储文件元数据（权限、大小等）及数据块指针。

- **扩展性**：支持子目录嵌套、符号链接等复杂逻辑。

（1）硬链接（Hard Link）

- **原理**：多个目录项指向同一inode（直接共享数据块）。
- **特点**：
  - 仅限同一文件系统内，无法跨设备。
  - 文件删除需所有硬链接均被删除（引用计数归零）。

（2）符号链接（Symbolic Link）

- **原理**：创建特殊文件，内容为目标路径字符串（间接引用）。
- **特点**：
  - 可跨文件系统，支持目录链接。
  - 目标删除后失效（悬空链接）。

（3）共享挂载（Mount）

- **原理**：将同一存储设备挂载到多个目录路径（如`/mnt`和`/backup`访问同一磁盘）。
- **特点**：适用于网络文件系统（NFS）或物理多路径访问。

（4）访问控制列表（ACL）

- **原理**：为不同用户/组设置细粒度权限（如`setfacl`命令）。
- **特点**：在标准权限（`rwx`）基础上扩展共享灵活性。

- 什么是**绝对路径与相对路径**？假设当前目录/home/wang/test，需要访问当前目录下的hello.cpp文件，写出绝对路径和相对路径

1. 定义

- **绝对路径**：从**根目录（/）**开始的完整路径，唯一确定文件位置。
- **相对路径**：从**当前目录**出发的路径，依赖工作目录的上下文。

2. 示例（当前目录：`/home/wang/test`，访问`hello.cpp`）

路径类型	示例路径	解析
绝对路径	`/home/wang/test/hello.cpp`	从根目录逐级定位到文件。
相对路径	`./hello.cpp` 或 `hello.cpp`	`./`表示当前目录，可省略。

3. 关键区别

特性	绝对路径	相对路径
基准点	根目录（/）	当前目录（`.`）
唯一性	是（全局唯一）	否（随工作目录变化）
使用场景	脚本、跨目录访问	当前目录操作（简化命令）

- *IO控制方式

> 程序直接控制（轮询）
>
> 中断驱动
>
> DMA直接存储器存取
>
> 通道控制

- *IO软件层次结构

> 用户层软件
>
> 设备独立性软件
>
> 设备驱动程序
>
> 中断处理程序

- 简述**磁盘空间分配**方法（文件的分配方法）

> 顺序分配
>
> 链接分配
>
> - 隐式
>
> - 显式 FAT
>
> 索引
>
> - 单级索引
> - 多级索引
> - 混合索引

- **磁盘调度策略**有哪几种，简要说明各磁盘调度算法的原理 +1 +1

> 1. **先来先服务（FCFS）**
>   - **原理**：按请求到达顺序依次处理。
>   - **特点**：公平但效率低，磁头移动距离大。
> 2. **最短寻道时间优先（SSTF）**
>   - **原理**：优先处理离当前磁头位置最近的请求。
>   - **特点**：减少寻道时间，但可能导致饥饿（远距离请求长期等待）。
> 3. **扫描算法（SCAN，电梯算法）**
>   - **原理**：磁头单向移动到底（如从内到外），途中处理请求，到达末端后反向移动。
>   - **特点**：无饥饿，但两端请求响应时间不均。
> 4. **循环扫描算法（C-SCAN）**
>   - **原理**：类似SCAN，但磁头到达一端后直接返回起点（不处理返回途中的请求）。
>   - **特点**：请求响应时间更均匀，适合负载均衡。
> 5. **LOOK与C-LOOK算法**
>   - **原理**：SCAN/C-SCAN的优化版，磁头移动只需到达最远请求位置即反向（无需到底）。
>   - **特点**：进一步减少无用移动，提升效率。

- 在**磁盘上进行一次读写操作需要哪几部分时间**，哪部分时间最长

1. 时间组成部分

1. **寻道时间（Seek Time）**
   - **定义**：磁头移动到目标磁道的时间。
2. **旋转延迟（Rotational Latency）**
   - **定义**：磁盘旋转到目标扇区的时间。
3. **传输时间（Transfer Time）**
   - **定义**：数据从磁盘读取或写入的时间。

2. 最长耗时部分

- **寻道时间**通常是主要瓶颈（尤其在随机访问场景），因其依赖机械移动。
- **旋转延迟**次之，而**传输时间**通常最短（除非传输大量连续数据）。

3. 总时间公式

$$T_{总} = T_{寻道} + T_{旋转延迟} + T_{传输}$$

- **最耗时部分**：**寻道时间**（机械移动不可并行）。
- **优化方向**：
  - 减少寻道（如磁盘调度算法：SCAN、SSTF）。
  - 降低旋转延迟（提高RPM或使用SSD消除机械部件）。

- 简述**硬中断与软中断**的概念，缺页中断与Ctrl+C分别属于什么中断

1. 硬中断（Hardware Interrupt）

- **触发方式**：由**硬件设备**（如键盘、磁盘、网卡）通过中断控制器（如APIC）向CPU发送信号。
- **特点**：
  - 异步发生，与当前执行的指令无关。
  - 需要保存现场，快速响应（如时钟中断）。

2. 软中断（Software Interrupt）

- **触发方式**：由**程序主动调用**（如`int 0x80`系统调用）或内核事件（如任务调度）引发。
- **特点**：
  - 同步执行，是程序流程的一部分。
  - 通常用于实现系统调用、异常处理等。

缺页中断 vs. Ctrl+C中断

中断类型	分类	触发原因	处理机制
缺页中断	**软中断**	访问的虚拟页面未加载到物理内存	操作系统调入缺失页，更新页表后重试指令

中断类型	分类	触发原因	处理机制
Ctrl+C	硬中断	用户按下中断键（SIGINT信号）	内核向目标进程发送信号，触发默认处理（终止进程）

关键结论

- **硬中断**：外部硬件触发，需快速响应（如设备I/O）。
- **软中断**：内部逻辑触发，同步处理（如缺页、系统调用）。
- **典型场景**：
  - **缺页中断**属于软中断（由CPU指令触发的同步异常）。
  - **Ctrl+C**属于硬中断（由键盘硬件触发的异步信号）。

- 叙述在**中断控制方式中输入请求I/O处理**的详细过程

1. **设备发起中断**：I/O设备完成操作后向CPU发送中断请求信号。
2. **CPU响应中断**：CPU保存当前上下文（PC、寄存器等），转入中断处理程序。
3. **执行中断服务程序（ISR）**：
   - 读取设备状态，确认中断原因。
   - 处理数据（如从设备缓冲区读取输入数据）。
   - 清除设备中断标志。
4. **恢复上下文**：CPU恢复之前保存的上下文，继续原任务。

关键得分点：中断触发→保存现场→ISR执行→恢复现场。

问答题（6*10=60）

- 页式系统地址转换 +1 +1 +1 +1
- 进程(作业)调度算法的等待时间和周转时间的分析以及**甘特图** +1 +1 +1 +1 +1 +1
- PV操作 +1 +1 +1 +1 +1 +1
- 理发师问题
- 银行家算法 +1 +1 +1
- 页面置换算法 +1 +1 +1 +1
- 按行存取与按列存取的数组占用页面数以及缺页中断数 +1
- 进程资源分配策略
- 磁盘调度算法原理以及时间计算 +1