# **OS复习提纲**

2025-05-31 chasetsai

## 一、概论

### **（一）操作系统的工作模式**

1. 用户程序库访问系统调用的过程是先执行一个trap指令陷入内核，然后跳转到具体的系统调用例程执行。()

【答案】bingo √，trap指令，同时切换CPU的状态

1. 操作系统在执行系统调用时可以被中断。（）

【答案】√

## 启动

启动流程：

1. BIOS完成硬件初始化
2. 然后从MBR（位于磁盘第0磁头第0磁道第1扇区）读取Bootloader
3. Bootloader stage1: 加载并跳转到stage2
4. Bootloader stage2: 加载操作系统内核到RAM

## 存储管理

### **编译链接原理**

#include <stdio.h>

int gdata1 = 10;

int gdata2 = 0;

int gdata3;

static int gdata4 = 11;

static int gdata5 = 0;

static int gdata6;

int main()

{

int a = 12; // 会放在text段,当作代码而非变量

int b = 0; // 运行时a,b,c放在stack

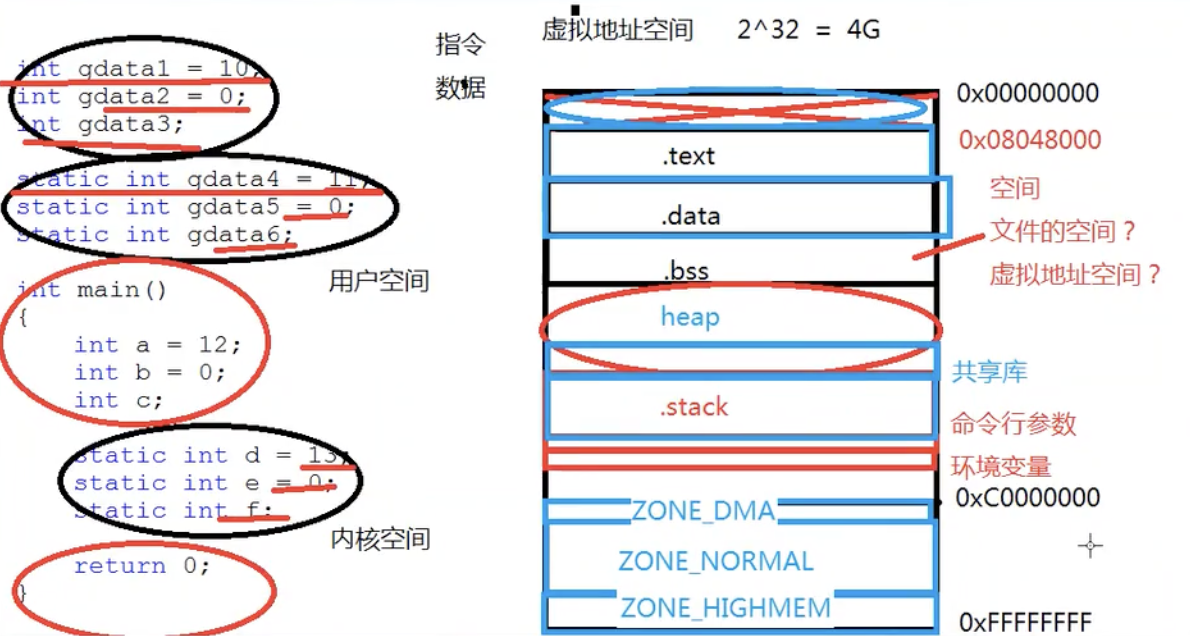
int c; //

static int d = 13; // data

static int e = 0; // bss

static int f; // bss

return 0;

}

每个程序运行的时候，操作系统会给其4GB（32位系统）的虚拟地址空间

- 初始化且不为0的全局变量放在data段

- text, data, bss 段是不可伸缩的

他存在，你看得见，他是物理的；

他存在，你看不见，他是透明的；

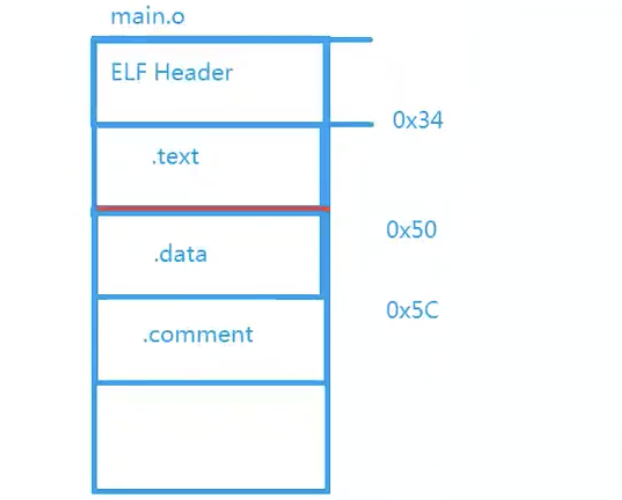
他不存在，你却看得见，他是虚拟的。

- #include <stdio.h> 只有函数的声明，未给出定义，定义在共享库内。编译C代码的时候，会自动链接libc.so (C的动态链接库/共享库)

**编译：**

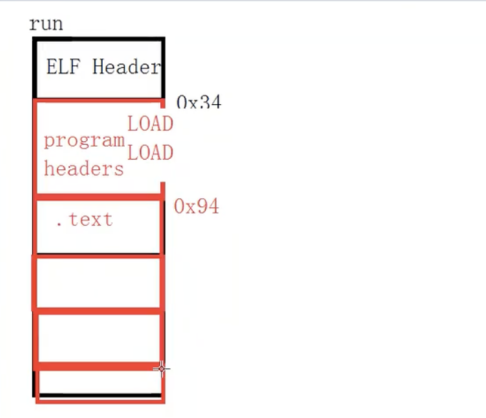
1. 预编译
   1. 进行一些宏的替换
   2. 得到 main.i
2. 编译
   1. 语法分析、代码优化、汇总所有符号，注意此时填充的地址均不正确（用的是不可访问的0x0）
   2. 得到 main.s
3. 汇编
   1. 把汇编指令转成机器码
   2. 得到 main.o (二进制可重定位目标文件)，并没有分配地址

**细看main.o**

- bss is short for “better saved space”，在.o文件中并不占空间

**链接：**

1. 合并所有obj文件的段，调整段偏移和段长度，合并符号表，进行符号解析 (即分配虚拟地址)。具体就是：
   1. 相同属性的段合并，组织在一个页面上 (不单单可以把各个text放一起，还要把所有只读的段都放一起，充分利用空间)
   2. 对所有obj文件的global符号进行处理，local (比如加了static的，表示只有本文件可见) 的符号不作任何处理
   3. 符号解析：所有obj符号表中对符号引用的地方要找到其定义的地方 (比如找到那些extern)
2. 给符号分配内存地址
3. **符号的重定位** (核心)：填上正确的地址，数据填绝对地址，函数填跳转的偏移
4. 得到可执行文件

**细看可执行文件**

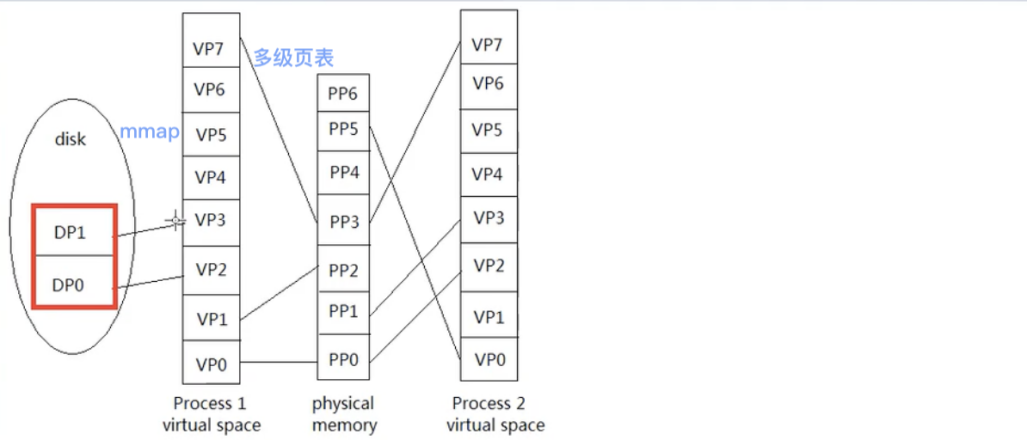
程序的运行步骤：

1. 创建虚拟地址空间到物理内存的映射，创建内核映射结构体，创建页目录和页表
2. 加载代码段和数据段
3. 把可执行文件的入口地址写到CPU的PC寄存器里

可执行文件比 .o 多了 program header，指示把每个段加载在哪里：

第一个LOAD页保存 .text 段，第二个LOAD页保存 .data 和 .bss

磁盘加载可执行文件：从磁盘用mmap映射到虚拟地址空间中，还要映射共享库



### **重定位**

名词解释——重定位

还要斟酌一下

所谓重定位是把作业的地址空间中的相对地址转换成内存空间的物理地址的调整过程。

在程序实际运行前，由操作系统把程序在内存的开始地址送入重定位寄存器；在程序运行期间，凡遇到访问内存的操作，就由硬件机制自动把用户程序的相对地址加上重定位寄存器的内容，相加之和就是实际访问内存的有效地址。

### **页式内存管理**

T1. 大小端：以0x12345678为例

大端模式

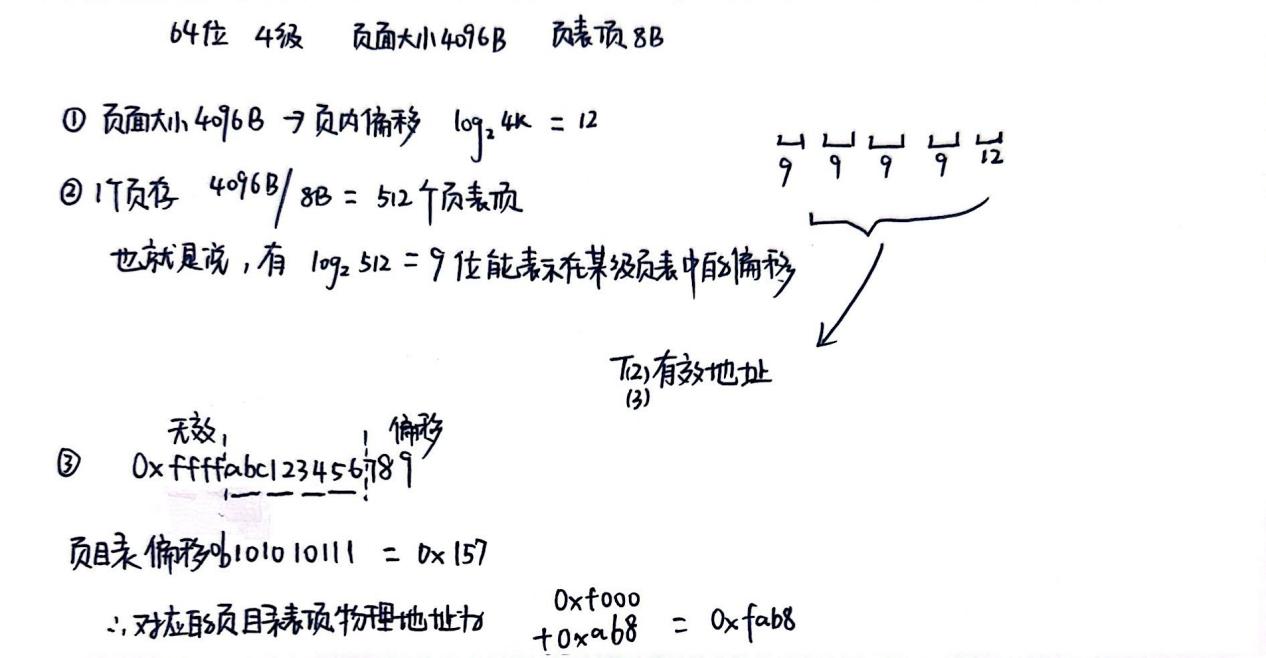
低地址 --------------------> 高地址  
0x12  |  0x34  |  0x56  |  0x78

T2. 非32位系统

一个64位系统架构中内存采用4级页表管理，每个页面大小4096字节，每个页表项占8字节

1. 页内偏移在页表项中占几位
2. 计算有效虚拟地址位长度和有效地址空间大小
3. 画出虚拟地址结构，标出各个部分起止位置和长度
4. 假设当前进程的页目录基址为0xf000，给出虚拟地址0xffffabc123456789对应的页目录表项的物理地址
5. 0，因为页表项不存页内偏移

(2)~(4)



### **段式内存管理**

1. 可重入

如果一个程序为多个进程所共享，那么该程序的代码在执行的过程中不能被修改，即该程序应该是 。

【答案】可重入的 (Reentrant code)

即只读、可被中断

【辨析】

一个函数被重入，表示这个函数没有执行完成，由于一些因素，又一次进入该函数执行。函数被重入只有两种情况：

1. 多个线程同时执行这个函数
2. 函数自身调用自身

可重入是指函数被重入后不会产生任何不良后果

## 进程管理

### **进程线程基本概念**

1. 进程上下文切换的过程（从A到B）
   1. 保存进程A的硬件上下文（即寄存器值），更新PCB（如状态改为就绪态），将进程A移入响应的队列（就绪or等待）
   2. 更新B的PCB为运行，加载B的上下文（恢复寄存器，切换地址空间）
   3. 执行进程B
2. 进程通信

进程通信的常用方式有 、 、 。

【答案】管道，共享内存，消息系统

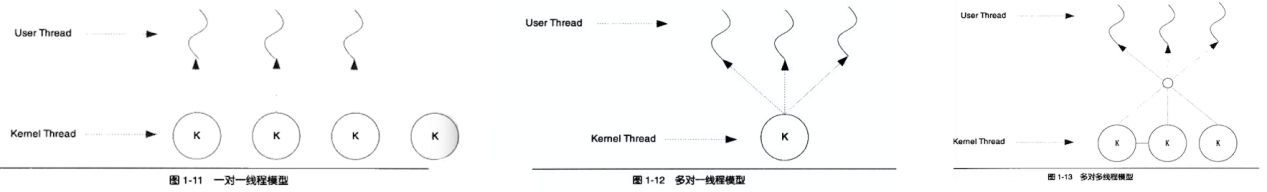
1. 管道

管道是 双工通信，要实现父子进程双方互动通信，需要定义两个管道。

【答案】半

1. **多线程内部情况——用户线程和内核线程**

内核线程由多处理器或调度来实现并发，而用户实际使用的线程并不是内核线程，而是存在于用户态的用户线程。

1. 一对一模型：一个用户使用的线程唯一对应一个内核使用的线程
2. 多对一模型：多个用户线程映射到一个内核线程上，线程之间的切换由用户态的代码来进行。缺点是如果其中一个用户态线程阻塞，那么所有的线程都将无法执行，而且无法充分利用多处理器的优势
3. 多对多模型：将多个用户线程映射到少数但不止一个内核线程上
4. 调度

若系统中只有用户级线程，则处理机调度单位是 。

【答案】进程，操作系统看不到用户级线程，只当作一个进程进行调度

### **同步与互斥**

1. 互斥区

互斥区管理遵循的四个原则？

【答案】空闲让进；忙则等待；有限等待；让权等待（进程不能进入临界区时，应立即释放处理机）

1. 忙等的硬件实现

* TestAndSet: 是不会被中断的原子指令，功能为写值 (true) 到某个内存位置并传回其旧值

1. 多进程同步barriers: 等所有进程都到达barrier后才开始执行

int count = 0;

semaphore mutex = 1;

semaphore barrier = 0;

P(mutex);

count++;

if (count == n) {

    V(barrier);

}

V(mutex);

P(barrier);

V(barrier);

1. 判断

用P、V操作可以解决一切互斥与同步问题。（）

【答案】√

1. 一个进程要向另一个进程传送大量数据，如不考虑进程间的同步，效率最高的进程间通讯机制是 。

【答案】共享内存

1. 管程：

将共享数据和操作这些数据的方法封装在一起

### **PV操作**

1. 生产者消费者

三个进程P1、P2、P3互斥使用一个包含N(N>0）个单元的缓冲区。P1每次用produce()生成一个正整数并用put()送入缓冲区某一个空单元中；P2每次用getodd()从该缓冲区中取出一个奇数并用countodd()统计奇数个数；P3每次用geteven()从该缓冲区中取出一个偶数并用counteven()统计偶数个数。请用信号量机制实现这三个进程的同步与互斥活动，并说明所定义的信号量的含义。要求用伪代码描述。

semaphore mutex = 1;    *// 缓冲区*

semaphore empty = N;

semaphore odd = 0;

semaphore even = 0;

P1() {

    P(empty);

    int num = produce();

    P(mutex);

    把num送入缓冲区...

    V(mutex);

    if (num & 1 == 1) {

        V(odd);

    } else {

        V(even);

    }

}

P2() {

    P(odd);

    P(mutex);

    getOdd();

    countOdd();

    V(mutex);

    V(empty);

}

1. 生产者消费者问题

一个完整机械产品由1个零件A和3个零件B装配完成。有5条生产线并行生产，其中1条生产零件A，3条生产零件B，1条负责装配产品。规定：

1. 每条零件生产线生产出来的零件**输入到装配生产线**。

2. 装配生产线可以接收最多1个零件A和3个零件B，此外没有其他额外的仓储空间，因此如果已经有1个零件A在**等待装配**，零件A的生产线就需要等待；同样，如果已经有3个零件B等待装配，则零件B生产线需要等待。

3. 每当装配生产线等到**接收到足够零件**后，就开始完成一个产品组装过程，之后激活等待的其他零件生产线。

使用P、V 操作给出上述过程中零件A、B生产线以及装配生产线的并发与同步控制过程，给出必要的注释。

semaphore aEmpty = 1;

semaphore aFull = 0;

semaphore bEmpty = 3;

semaphore bFull = 0;

semaphore mutex = 1;

producerA() {

    P(aEmpty);

    生产1个零件A...

    P(mutex);

    把生产出的零件送到装配处...

    V(mutex);

    V(aFull);

}

producerB() {

    P(bEmpty);

    生产1个零件B...

    P(mutex);

    把生产出的零件送到装配处...

    V(mutex);

    V(bFull);

}

assembler() {

    P(aFull);

    P(bFull);

    P(bFull);

    P(bFull);

    装配零件...

    V(aEmpty);

    V(bEmpty);

    V(bEmpty);

    V(bEmpty);

}

1. 校车问题 [难难难]

乘客来到校车的停车站等待校车。当巴士到达的时候，所有正在等待的乘客调用boardBus()上车。一旦开始上车，任何**新到来的乘客都必须等待下一辆**巴士。校车的容量为50人，如果有**超过50个人排队，50名之外的乘客需要等待下一辆巴士**。当所有等待的乘客**上车完毕，巴士可以离开**（调用depart）。如果巴士到达时，没有任何乘客，它就会立刻离开。请用PV操作编写巴士进程和乘客进程的同步互斥关系。

【不知道对不对哈】

看到的答案都不能保证“开始上车后，新到的乘客必须等下一辆巴士”，不知道是不是我没get到别人的代码

1. 信号量

假设有n个旅客和1辆汽车，旅客在汽车停靠的站点反复乘车，汽车一次可以乘坐C个旅客(C<n)。汽车在**坐满C个旅客后出发绕一圈**，回到原来站点**让旅客下车**。旅客和汽车重复这个过程，注意二者需要满足下列条件：

旅客能够上车和下车；汽车能够载客、运行和卸客。

只有汽车处于载客状态后，旅客才可以上车。只有C个旅客上车后，汽车才可以出发运行。只有汽车处于卸客状态后，旅客才可以下车。只有**旅客都下车后，汽车才能重新载客**。

请用P、V操作实现旅客和汽车之间的同步关系。

int count = 0;

semaphore mutex = 1;

semaphore seat = C;

semaphore full = 0;     *// 车满*

semaphore empty = 0;    *// 车空*

semaphore load = 0;     *// 上车*

semaphore unload = 0;   *// 下车*

passenger() {

*// 上车*

    P(load);    *// 等待车准备好*

    P(seat);

    P(mutex);

    count++;

    if (count == C) {

        V(full);    *// 提醒发车*

    } else {

        V(load);    *// 提醒下一个乘客上车*

    }

    V(mutex);

*// 下车*

    P(unload);

    P(mutex);

    count--;

    if (count == 0) {

        V(empty);   *// 提醒车辆已空*

    } else {

        V(unload);  *// 提醒其他乘客下车*

    }

    V(mutex);

    V(seat);

}

car() {

*// 载客*

    V(load);    *// 告诉第1个乘客上车*

    P(full);

    wandering();

*// 卸客*

    V(unload);

    P(empty);

}

1. 读者写者问题

某条河上只有一座独木桥，以便行人过河。现在河的两边都有人要过桥，按照下面的规则过桥，为了保证过桥安全，请用P、V操作分别实现正确的管理。过桥的规则是：同一方向的可连续过桥，某方向有人过桥时另一方向的人要等待。

int cnt1 = 0;

int cnt2 = 0;

semaphore mutex1 = 1;

semaphore mutex2 = 1;

semaphore bridge = 1;   *//桥*

edge1() {

    P(mutex1);

    if (cnt1 == 0) {

        P(bridge);      *// 第一个负责抢桥,如果失败,也不会释放mutex1,之后这边的车辆就会被阻塞*

    }

    cnt1++;

    V(mutex1);

    goAhead();

    P(mutex1);

    cnt1--;

    if (cnt1 == 0) {

        V(bridge);

    }

    V(mutex1);

}

1. 读者写者

设有三个并发进程：进程Reader负责从输入设备**读入信息并传送给进程Handler**，进程Handler将**信息加工并传送给进程Printer**，进程Printer将进行打印输出。其中，三个进程**共享同一个缓冲区**，且缓冲区**大小为K**。请使用P/V操作，写出正确的并发程序。

semaphore empty = k;

semaphore h\_ok = 0;

semaphore p\_ok = 0;

semaphore mutex = 1;

Reader() {

    P(empty);

    readData();

    P(mutex);

    putDataToBuffer();

    V(mutex);

    V(h\_ok);

}

Handler() {

    P(h\_ok);

    P(mutex);

    handleBufferData();

    V(mutex);

    V(p\_ok);

}

Printer() {

    P(p\_ok);

    P(mutex);

    printBufferData();

    V(mutex);

    V(empty);

}

### **进程调度**

1. 周转时间：完成时刻-到达时刻

带权周转时间：周转时间/执行时间

1. 进程因从网络设备读取数据包而阻塞，然后调度程序将该进程换出。这是一个自愿切换上下文的例子。（）

【答案】√

时间片到才是非自愿

### 截屏2025-06-01 20.04.18**死锁**

1. 银行家算法

银行家算法是在保证至少有一个进程能得到所需的全部资源的前提下进行资源分配的。（）

【答案】√

## I/O

1. 外设分类

磁盘是一类典型的字符设备。（）

【答案】×

* 信息交换单位：块设备和字符设备
* 块设备：以块为传输单位的设备，每个块可以独立于其他块进行读写。例如：硬盘、CD-ROM、USB盘等
* 字符设备：以字符为单位发送或者接收字符流的设备，不可寻址。例如：打印机、网卡、鼠标等

1. 设备独立性

设备独立性（或无关性）是指能独立实现设备共享的一种特性。（）

【答案】×

设备独立性是指用户在编制程序时所使用的设备与实际使用的设备无关

程序可以通过一组统一的操作来操作设备，这种操作的接口与具体设备无关

1. 用户使用设备名

用户程序使用 设备名，系统实际执行时，映射到物理设备名。

【答案】逻辑

1. 与设备相关的中断处理过程由设备驱动程序完成。（）

【答案】√ ，中断处理程序是设备驱动程序的三大组成部分之一

1. 中断与DMA比较

中断方式的数据传送是在中断处理时由CPU控制完成的；DMA方式则不经过CPU，而是在DMA控制器的控制下完成的。（）

【答案】√

1. 在常用的IO控制方式中，要求主存与IO设备之间有直接数据通路的方式是 。

【答案】DMA (Direct Memory Access)

I/O通道：在执行数据传输时，也是通过直接访问内存来实现的，这与DMA的原理是一致的。但我们的核心关注点是通过**可编程的通道**来管理复杂的I/O序列和多设备操作，进一步减轻CPU负担

1. I/O软件层

以下各项任务是在四个I/O软件层的哪一层完成的？

1. 为一个磁盘读操作计算磁道、扇区、磁头 驱动
2. 向设备寄存器写命令 驱动
3. 检查用户是否允许使用设备 设备无关软件
4. 将二进制整数转换成ASCII码以便打印 用户
5. 将系统调用参数翻译成设备操作命令 设备无关软件

【分析】四个软件层：用户层软件，设备独立性软件，设备驱动程序，中断处理程序

1. 设备控制器

设备控制器是一块能控制一台或多台外围设备与CPU并行工作的硬件。（）

【答案】√

## 磁盘管理

1. **磁盘调度**
2. 磁壁粘着现象：少数进程高频访问某个磁道导致其他请求饥饿，采用N-Step-SCAN可以避免
3. **RAID**

## 文件系统

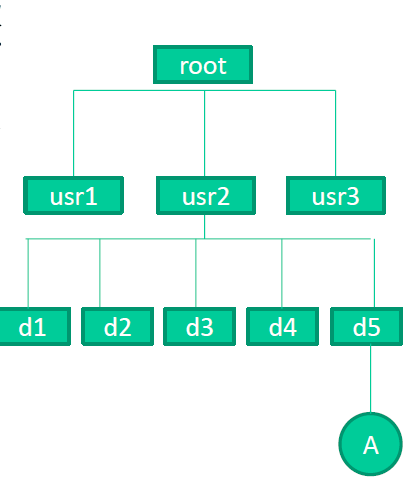
1. 文件系统的作用

操作系统在文件管理方面有哪五项主要活动

【答案】目录管理、文件存储空间管理、文件共享、文件保护、数据一致性控制

1. 文件结构

某文件系统以硬盘作为文件存储器，物理块大小为512B。有文件A包括**490个逻辑记录**，每个记录占255B，**每个物理块存2个记录**。文件A在该文件目录中的位置如图所示。每个目录项占127B，**每个物理块存放4个目录项**。**根目录内容常驻内存**。



若文件采用链接分配方式，如果要将文件A读入内存，至少要存取几次硬盘？

若文件采用连续分配方式，如果要读取逻辑记录号为480的记录，至少要存取几次硬盘？

【答案】

串联式文件：

根目录的内容是目录项，找到usr2的目录项，访问 1~2 次磁盘找到含有d5目录项的磁盘，访问d5目录项包含的磁盘块号对应的磁盘块 1 次，读出A包含的磁盘块号，文件A有490个记录，所以共占245个物理块，访问245次磁盘读出。所以至少 1+1+245 次

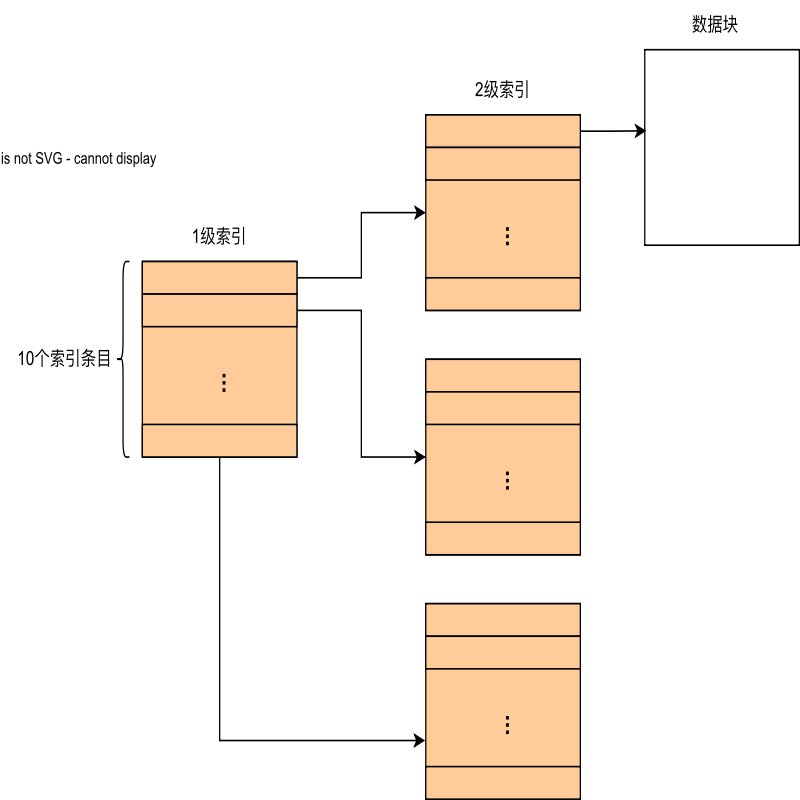
连续分配：

读出A包含的磁盘块号后，直接找到第480个记录对应的磁盘块号，访问 1 次磁盘就能读到。至少共 1+1+1=3 次

1. 索引文件

一个包含多个记录的索引文件，**每个记录正好占用一个物理块**。如果每**一个物理块可以存放10个索引表目**，在建立多级索引结构时，**一个物理块需要对应一个索引表目**，每级索引至少占用一个物理块，则存储100个记录，至少需要占用 个索引块。

【答案】

用2级索引，至少11块

1. 文件系统

一个文件系统采用多级索引结构管理物理存储块，其文件控制块中包含10个直接索引、**1个一次间接索引**、**1个二次间接索引**。文件的每个物理存储块大小为512B，存储块**索引采用32位**指针。

1. 请问该文件系统是否支持10MB文件的管理，为什么？
2. 如果一个文件大小为1MB，请计算存储该文件所需物理存储块数量（不考虑文件控制块所占空间）。
3. 如果该文件系统主要用于存储大量**100KB**大小的文件。请在上述文件索引结构基础上设计一个更加高效的索引结构。注意，文件控制块中用于存放物理存储块索引指针的空间不可扩充，单个文件的最大容量**不得降低**。

【答案】

1. 一个物理块可以存 512B/4B=128 个索引，一个文件最多包含 10+128+128\*128 个磁盘块，文件最大为 (10+128+128\*128)\*512B < 10MB，所以不支持
2. 10个1级索引可以表示的文件大小：10\*512B=5KB

2级索引可以表示的文件大小：128\*512B=64KB

所以还需要3级索引，去指示剩下的 (1MB-5KB-64KB)/512B=1910 块，1910/128=14.9

故需索引块 1+1+1+15=18，数据块 1MB/512B=2048，一共2066块

1. 100KB/512B=200块

更加高效 → 减少磁盘访问次数

2个二级索引就能满足200块，所以索引结构：9个直接索引，2个一级索引，1个二级索引