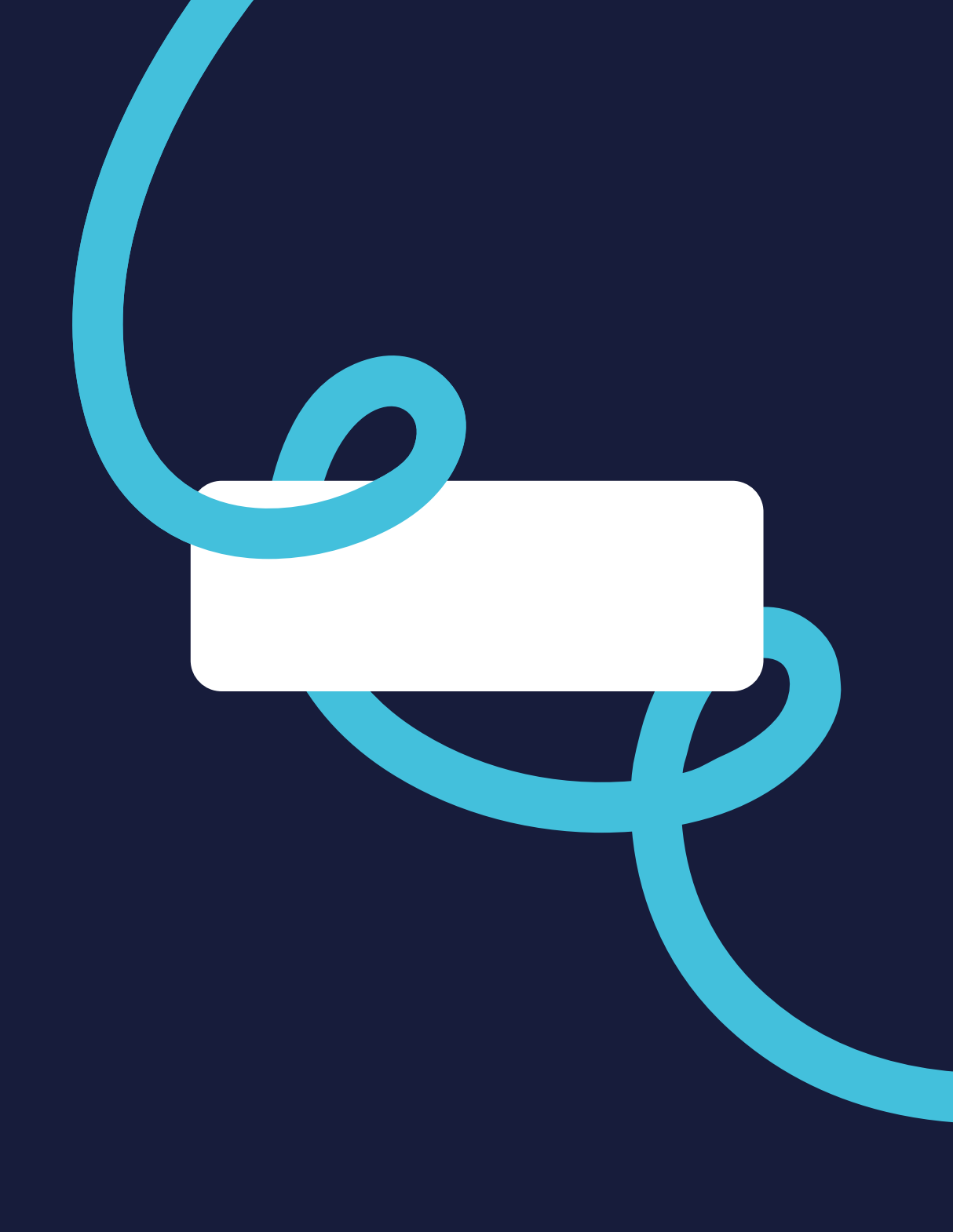


操作系统(OS)：是管理计算机硬件和软件资源的系统软件
是用户和计算机硬件之间的接口

四个主要管理模块：处理器管理、存储器管理、设备管理、文件管理

四个基本目标：方便性，有效性，可扩充性，开放性

四个基本特性：并发性 ↓ 系统中同时存在多个运行中的程序
共享性 ↓ 系统资源可被多个进程同时使用
虚拟性 ↓ 把物理资源变为逻辑上更多的资源（虚拟内存）
不确定性 ↓ 进程执行顺序和时间不可预知

三种基本类型：① 批处理操作系统：作业成批地提交，自动运行无交互，吞吐量大
② 实时操作系统：对外部事件及时响应可靠性要求极高
③ 分时操作系统：多用户共享CPU，时间片轮转有交互

操作系统用户接口：程序接口（系统调用）和 操作接口（命令行/图形界面）

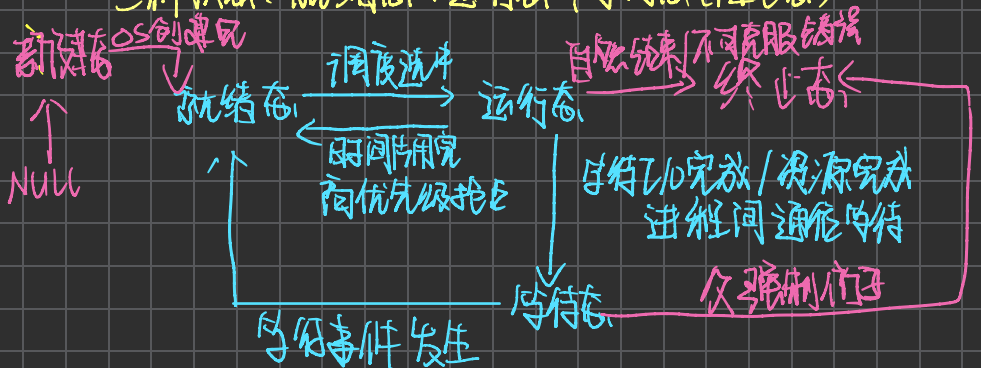
物联网四个逻辑层：① 传感层 ② 传输层 ③ 处理层 ④ 应用层

其他OS（了解）：网络OS：客户机/服务器模式，资源共享
分布式OS：总体耦合，对用户透明
嵌入式OS：微型化，实时，可靠

进程：进程是具有力

进程三态转换：

三种状态：就绪态、运行态、等待态（阻塞态）



P.S：等待不能直接变运行态，必须先回就绪态

进程五态：

新增

新建态：OS已创建PCB，分配了进程ID，但尚未进入内存

终止态：进程运行结束，不再有执行资格，等待OS做善后

NULL → 新建态：创建一个新进程

新建态 → 就绪态：OS完成创建且内存/性能允许，进程进入就绪态

运行态 → 终止态：进程到达自然结束点，出现不可克服的错误

终止态 → NULL：完成善后操作后，OS在PCB删除

就绪/等待 → 终止态：某些OS允许父进程强制终止子进程

PCB（进程控制块）

系统为管理进程设置的数据结构，记录进程外部特征和运动变化过程

有：进程ID，进程状态，CPU寄存器，内存信息，IO状态

进程 vs 程序

五、进程 vs 程序的区别（简答常考）

对比项	进程	程序
动/静	动态（有生命周期）	静态（文件）
存在时间	临时	相对长久
组成	程序 + 数据 + PCB	指令序列
对应关系	一个程序可对应多个进程	一个程序可被多个进程

进程三要素：程序，数据和栈，PCB

Day 2

死锁：多个进程因竞争资源（系统拥有的资源数量有限，或并发进程推进顺序不当）而进入的一种永远等待资源的僵局

死锁 vs 饥饿（习题册原题）：

	死锁	饥饿
本质	永远等待资源的僵局	长时间得不到资源
原因	资源竞争 + 推进顺序不当	分配策略不公平
系统状态	多个进程相互阻塞	单个进程被无限延迟
能否结束	不能自行结束	可能因策略调整结束

导致死锁的4个必要条件：

①互斥条件：资源在一个时间内只能被一个进程使用

②请求和保持条件：进程已经有至少一个资源，又请求新资源被阻塞，但不释放已占资源

③不剥夺条件：进程已获得的资源在未使用完前不能被强行剥夺

④环路等待条件：存在一个进程-资源的循环等待链

解决死锁的几种方法

① 预防死锁（最严格）：在系统设计时就破坏4个必要条件之一，从根本上杜绝死锁

② 避免死锁（较严格）：允许4个条件存在，但在资源分配时动态地判断是否进入不安全状态（银行家算法就是这一类）

③ 检测死锁（放宽）：允许死锁发生，定期运行检测算法，发现死锁后处理

④ 解除死锁（配合检测）：检测到死锁后，撤销某些进程或剥夺其资源来解锁

死锁预防的具体做法——逐条破坏

不破坏"请求和保持"：一次性分配（破坏"请求"）：进程运行前一次性申请所有资源，否则不允许运行

不破坏"不剥夺"：进程申请新资源若不能立即获得则释放已占有的全部资源，以后重新申请

不破坏"环路等待"：有序资源分配：对所有资源统一编号，进程必须按编号递增顺序申请

五、银行家算法（必考·大题）

5.1 名字由来

银行家算法是 Dijkstra 提出的，类比银行家给客户放贷：只有保证所有客户都能完成贷款时，才放出当前请求的款项。

5.2 四个数据结构（必背）

设系统有 **n** 个进程、**m** 种资源：

名称	含义
Available[m]	系统当前**可用资源向量**（每种资源还剩多少）
Max[n,m]	进程 Pi 对每类资源的**最大需求**
Allocation[n,m]	进程 Pi **已分配**到的资源数
Need[n,m]	进程 Pi **还需要**的资源数

关键公式：`Need[i,j] = Max[i,j] - Allocation[i,j]`

阶段 A：请求处理

操作… 2 artifacts

当进程 Pi 提出请求向量 `Request[i]` 时：

1. 若 `Request[i] ≤ Need[i]`？否→报错（超出申报）
2. 若 `Request[i] ≤ Available`？否→等待
3. 试分配：
   - `Available = Available - Request[i]`
   - `Allocation[i] = Allocation[i] + Request[i]`
   - `Need[i] = Need[i] - Request[i]`
4. 跑安全性算法→安全则真分配，不安全则回滚让 Pi 等待

阶段 B：安全性算法（找安全序列）

1. 设 `Work = Available`，`Finish[i] = false`（所有 i）
2. 找一个满足 `Finish[i]=false ∧ Need[i] ≤ Work` 的进程 Pi
3. 若找到：让它执行完归还资源→`Work = Work + Allocation[i]`，`Finish[i] = true`
4. 重复 2-3 直到所有 `Finish[i]=true`（**安全**，输出序列）或找不到（**不安全**）

① 请求 < Need
② 请求 < 可用
③ 试分配
可用 = 可用 − 请求
已分配 = 已分配 + 请求
Need = Need − 请求
④
1. Work = 可用
2. Work > need?
3. 执行完归还
Work = Work + 分配
4. 到全了

5.4 速记套路（做题三步走）

> **算 Need → 检查 Request → 找安全序列**

做大题时按这个表格画就稳：

```
进程  | Allocation | Need  | Available（动态）
------|------------|-------|---------------
P0    |  ...       |  ...  |
P1    |  ...       |  ...  |
P2    |  ...       |  ...  |
```

然后逐一试：哪个进程的 Need ≤ Available？让它跑完，更新 Available，继续找下一个。能全部完成 → 安全序列存在 → 系统安全。

5.5 考点提示

- **选择题考过**：银行家算法的作用是 → **死锁避免**（不是预防、不是检测）。
- **大题考法**：给定 Allocation/Max/Available 三张表，问：
  - ① 此刻系统是否安全？给出安全序列。
  - ② 若 Pi 请求 Request=(x,y,z)，能否分配？

Day 3

一、同步与互斥

互斥（间接作用）：多个进程竞争使用同一个临界资源，同一时刻只能有一个进程使用

同步（直接作用）：多个进程为完成共同任务，协调彼此的执行顺序（一个进程告诉另一个进程结果）

（抢资源 -> 互斥；等结果 -> 同步）互斥是同步的一种特殊情况

临界资源：一次只允许一个进程使用的公共资源（如打印机）

临界区：进程中访问临界资源的那段代码

二、信号量与PV操作

信号量定义（课件原文）

信号量是一个数据结构，由 Dijkstra（1965）提出：

```
struct semaphore {
    int value;              // 信号量的值
    pointer PCBqueue;       // 信号量队列指针
}
```

P 操作和 V 操作（原语，不可中断）

申请资源、释放资源

```
P(s) {
    s.value --;
    if (s.value < 0) {
        进程进入等待队列;
        阻塞;
    }
}
```

```
V(s) {
    s.value ++;
    if (s.value ≤ 0) {
      从等待队列唤醒一个进程;
        该进程进入就绪队列;
    }
}
```

信号量的物理含义：

S>0　表示还有 S 个资源可用

S=0　资源刚好用完

S<0　有进程在为得该资源阻塞

PV 操作使用规则（必考）

1. 信号量必须置一次且只能置一次初值，且初值不能为负数
2. P、V 必须成对出现，有一个 P 就必有一个 V
3. 互斥时：P、V 在同一个进程中
4. 同步时：P、V 在不同进程中
5. P(S1) 和 P(S2) 一起用时：同步P 要在 互斥P 之前（否则可能死锁）

三、四大经典模板（必会）

1. 最简单的互斥

（多个进程访问同一个临界资源）

模板1：最简单的互斥 ★★

场景：多个进程访问同一个临界资源。

```c
semaphore mutex = 1;      // 初值为1
cobegin
  process Pi:
    begin
      ...
      P(mutex);           // 申请进入
      临界区;             // 访问临界资源
      V(mutex);           // 释放
      ...
    end;
coend;
```

semaphore mutex=1
cobegin
process Pi
begin
...
P(mutex)
临界区
V(mutex)
...
end
coend

信号量名：mutex
初值 =1
P, V包夹临界区且在同一进程

2. 最简单的同步

场景：进程 P2 必须等 P1 执行完某操作后才能继续。

```c
semaphore s = 0;          // 初值为 0（关键！）
cobegin
  process P1:                 process P2:
    begin                       begin
      ...                         P(s);        // 等 P1 的 V
      执行操作 A;                  执行操作 B;  // 必须在 A 之后
      V(s);   // 发信号          ...
      ...                         end;
    end;
coend;
```

同步信号量初值=0（互斥=1）

V在先做的进程，P在后做的进程

P，V不在同一进程

3、生产者-消费者问题

初值	
互斥信号量：①empty	K	空缓冲区数量（同步）
②full	0	满缓冲区数量（同步）
③mutex	1	缓冲区互斥访问（互斥）

```
var B: array[0..k-1] of item;
    in, out: integer := 0;
    empty, full, mutex: semaphore;
    empty := k;  full := 0;  mutex := 1;

cobegin
 process producer_i:                process consumer_j:
  begin                              begin
    L1: 生产一个产品;                  L2: P(full);     // 同步P
        P(empty);   // 同步P             P(mutex);    // 互斥P
        P(mutex);   // 互斥P             product := B[out];
        B[in] := product;               out := (out+1) mod k;
        in := (in+1) mod k;               V(mutex);
        V(mutex);                       V(empty);
        V(full);                      消费产品;
        goto L1;                        goto L2;
  end;                             end;
coend;
```

4. 读者-写者问题

(多个读进程和写进程共享一个文件)

读 vs 读：允许同时

读 vs 写：互斥

写 vs 写：互斥

信号量与变量

W	1	控制读写互斥（信号量）
R	1	控制rc的互斥访问（信号量）
rc	0	当前正在读的进程数（普通变量）

完整代码（课件原文）：

```c
var rc: integer := 0;
    W, R: semaphore;  W := 1;  R := 1;

cobegin
  procedure read:                          // 读者
    begin
      P(R);
      rc := rc + 1;
      if rc = 1 then P(W);
      V(R);
      读文件;
      P(R);
      rc := rc - 1;
      if rc = 0 then V(W);
      V(R);
    end;
  procedure write:                         // 写者
    begin
      P(W);
       写文件;
      V(W);
    end;
coend;
```

Day 4

PV 操作变体应用

一、苹果桔子问题

场景

桌上有一只盘子（每次只能放一个水果）：

- 父亲：往盘子放苹果
- 母亲：往盘子放桔子
- 女儿：等着吃苹果
- 儿子：等着吃桔子

这是什么类型?

生产者消费者的变体——但有特殊性：

- 父亲放苹果，只有女儿能吃（不是任意消费者）
- 母亲放桔子，只有儿子能吃
- 父母都要抢同一个盘子（盘子只放一个水果）

所以不能用 full、empty 两个信号量了，因为放苹果和放桔子要区分。

信号量设置

信号量	初值	含义
S[7]	1	盘子允许放入水果数（互斥兼空盘，初值表示有1个空位）
Sg1	0	盘子里桔子的数量（同步给儿子）
Sg2	0	盘子里苹果的数量（同步给女儿）

（SP：互斥+empty合二为一）

```
sp, sg1, sg2: semaphore;
sp := 1;  sg1 := 0;  sg2 := 0;

cobegin
  process father:
    begin
      L1: 削一个苹果;
          P(sp);     // 抢盘子
          放苹果;
          V(sg2);    // 通知女儿
          goto L1;
    end;

  process mother:
    begin
      L2: 剥一个桔子;
          P(sp);     // 抢盘子
          放桔子;
          V(sg1);    // 通知儿子
          goto L2;
    end;
coend;
```

```
process daughter:
  begin
    L4: P(sg2);   // 等苹果
        取苹果;
        V(sp);    // 还盘子
        吃苹果;
        goto L4;
  end;

process son:
  begin
    L3: P(sg1);   // 等桔子
        取桔子;
        V(sp);    // 还盘子
        吃桔子;
        goto L3;
```

二、变体延伸：父亲女儿2人版

考试如果只问**父亲放苹果、女儿吃苹果**（去掉母亲儿子），只需2个信号量：

Code

```
sp := 1;   // 盘子空位
sg := 0;   // 盘子里苹果数

process father:
  L1: 削苹果;
      P(sp);
      放苹果;
      V(sg);
      goto L1;
```

```
process daughter:
  L2: P(sg);
      取苹果;
      V(sp);
      吃苹果;
      goto L2;
```

> **考点提示**：父子型题目还有变体——“父亲放饭、母亲放菜、孩子吃饭和菜”，思路同苹果桔子，**每种食物配一个同步信号量。**

哲学家进餐问题

操作… 4 artifacts

场景

5个哲学家围坐圆桌，桌中央有一盘面，每人面前一只盘子，两人之间一把叉子（共5把叉子）。每个哲学家要同时拿到左右两把叉子才能吃面。

简单解法（先看会出问题的版本）

Code

```
fork[0..4]: semaphore;      // 5 把叉子
fork[i] := 1 for all i;

process Pi:  // i = 0,1,2,3,4
  begin
    L1: 思考;
        P(fork[i]);                 // 拿左边叉子
        P(fork[(i+1) mod 5]);       // 拿右边叉子
        吃面;
        V(fork[i]);                 // 还左边
        V(fork[(i+1) mod 5]);       // 还右边
        goto L1;
  end;
```

这个解法会死锁（必考点）

5个哲学家同时拿起左边叉子，然后都当右边一但右边用对也被拿着

解决方法

① 至多允许4个哲学家同用吃　　原理 限制并发数，破坏循环等待.

② 奇数号先拿左，偶数号先拿右　　打破对称性，避免都抢右

③ 两把叉子都能拿到才拿，否则一把也不取　　一次分配，破坏"请求和保持"

四、PV 题做题套路（关键）

碰到任何 PV 应用题，按以下 4 步走：

Step 1：识别有几"类"进程

数一数有几种角色（生产者？消费者？读者？写者？哪种特殊角色？）。

Step 2：找出资源关系

关系	信号量类型	初值
多人抢同一东西	互斥 mutex	1
A 等 B 完成	同步 s	0
缓冲区有 K 个空位	empty	K
缓冲区初始有 K 个产品	full	0（一般缓冲区开始为

Step 3：找"对子"

每个"等待-通知"关系，是一对 P-V：

- 等的一方写 P
- 通知的一方写 V

Step 4：检查规则

- 同步 P 在互斥 P 之前
- V 顺序无所谓
- P/V 数量配对

进程通信概念

1. 进程间通信定义

进程之间互相交换信息的工作。分两种：

- **低级通信**：传递控制信息（如信号量、PV操作）
- **高级通信**：大批量数据传送（Send/Receive 原语）

2. 四种通信方式（必背） ☆

方式	关系	特征
主从式	主进程-从进程	关系固定，从进程受主
会话式	使用进程-服务进程	使用方必须先获得许可
消息队列/邮箱	发送进程-接收进程	双方地位平等，靠缓冲
共享存储区	读进程-写进程	同一内存区域共享，通

控制（如终端控制）
（如客户-服务器）
飞传递
信速度最高

直接通信 VS 间接通信

直接通信：消息直接传给对方（指定对象）

间接通信：借助（ ）将数据结构（ ）作中转

邮箱，消息队列

4. Linux 的 3 种通信机制（了解）

- **信号通信**：软中断，传简单事件
- **共享存储区**：通信速度最高（shmget、shmat、shmc shmctl）
- **消息通信**：格式化消息队列

考点提示

- “通信速度最快的是？”→ **共享存储区**
- “信号通信又叫？”→ **软中断**
- “邮箱属于哪种通信？”→ **间接通信**

Day 5

一、三级调度

高级调度　　作业调度/宏观调度　　后备作业是否进入内存　分时级

中级调度　　平衡负载调度　　进程是否换出/换入主存

低级调度　　进程调度/短程调度　　就绪队列中谁占用CPU　更新

二、抢占式 vs 非抢占式

非抢占式：进程一旦占用CPU运行到结束或主动阻塞为止

抢占式：高优先级可强行夺取

三、五大调度算法（手算大题必考）

1. FCFS — 先来先服务

规则：按到达时间排序，谁先到谁先跑（非抢占）。

特点：

- ✅ 公平，简单
- ❌ 有利于长作业，不利于短作业
- ❌ 有利于CPU密集型，不利于I/O密集型

2. SJF — 最短作业优先

规则：在已到达的作业中，选估计运行时间最短的（非抢占）

特点：

- ✅ 平均周转时间最短
- ❌ 长作业可能"饥饿"（永远没机会跑）

3. SRTF — 最短剩余时间优先

规则：SJF的抢占版。新作业到达时，若它的运行时间比当前作业剩余时间还短，就抢占。

4. HRRN — 最高响应比优先 ★★

核心公式（必背）：

```
响应比 R = （等待时间 + 处理时间） / 处理时间
         = 1 + 等待时间 / 处理时间
```

规则：每次调度时，计算所有等待作业的响应比，选最大的运行（非抢占）。

优势：

- 短作业→处理时间小→响应比大→优先（兼顾SJF优势）
- 长作业等久→等待时间大→响应比也升高→不会饿死（兼顾FCFS公平）
- 是 FCFS 和 SJF 的折中（习题册原话）

5. HPF — 优先数调度

规则：按优先数高低调度。可静态/可动态、可抢占/可不抢占。

例题里的优先数公式（课件原文）：

```
优先数 = 用户规定优先数 - 作业处理时间 + 作业等待时间 - 输出
```

Opus 4.7

四、衡量指标（必背公式）

操作... 6 artifacts

设作业 i：

- 到达时间 = Si（进入输入井时间）
- 完成时间 = Ei（结果出来时间）
- 运行时间 = ri（实际执行需要的CPU时间）

指标	公式
周转时间	Ti = Ei − Si
带权周转时间	Wi = Ti / ri
平均周转时间	T = (∑Ti) / n
平均带权周转时间	W = (∑Wi) / n

> 💡 理解：
>
> - 周转时间 = 从提交到完成的总时间（包括等待+运行）
> - 带权周转时间 = 周转时间 / 运行时间（一定 ≥ 1）
> - 带权越接近 1，调度越高

手算流程

① 列出原始信息（到达时间、运行时间）

② 按算法规则确定执行顺序

③ 算每个作业的开始时间、完成时间

④ 算周转时间：完成 − 到达 （$T_i = S_i - E_i$）

⑤ 算带权周转：$= \frac{T_i}{r_i}$ $\frac{\text{周转}}{\text{运行}}$

⑥ 算平均

六、HRRN 详细做题步骤（最容易错）

HRRN比其他算法多一步：**每次调度时都要重新计算响应比。**

例子：3个作业，到达时间 8.8、9.0、9.5，运行时间 1.5、0.4、1.0。系统规定全部到达后开始调度。

Step 1：9.5时刻，3个作业都到了，第一次调度：

```
Code

作业1: R = 1 + (9.5-8.8)/1.5 = 1 + 0.47 = 1.47
作业2: R = 1 + (9.5-9.0)/0.4 = 1 + 1.25 = 2.25
作业3: R = 1 + (9.5-9.5)/1.0 = 1 + 0 = 1.00
→ 作业2 响应比最高，先跑
```

Step 2：9.9时刻（作业2跑完），剩下1和3，重算：

```
Code

作业1: R = 1 + (9.9-8.8)/1.5 = 1 + 0.73 = 1.73
作业3: R = 1 + (9.9-9.5)/1.0 = 1 + 0.4 = 1.40
→ 作业1 响应比最高，第二个跑
```

Step 3：作业1跑完(11.4)，剩作业3。3跑到12.4。

作业	到达	运行	开始	完成	周转
1	8.8	1.5	9.9	11.4	2.6
2	9.0	0.4	9.5	9.9	0.9
3	9.5	1.0	11.4	12.4	2.9

平均周转 = (2.6+0.9+2.9)/3 = **2.13**

> ⚠️ **HRRN的关键**：每次"上一个作业跑完时"重新计算响应比，不是开始时算一次就够了。

Day 6
存储管理基础

一、为什么要存储管理

存储管理的四大功能（了解）：内存分配与回收、地址转换、内存扩充、内存保护。其中**地址转换**是考试大题核心。

一个关键概念——**逻辑地址 vs 物理地址**：

- **逻辑地址**（相对地址）：程序里用的地址，从 0 开始编
- **物理地址**（绝对地址）：内存里真实的地址
- **地址转换**（重定位）：把逻辑地址换算成物理地址

二、可变分区 + 空闲区分配算法

可变分区是什么

不预先划分内存，作业来了**按需要的大小**动态切一块给它。久而久之内存里出现很多**空闲区**（碎片），于是需要算法决定"新作业放进哪个空闲区"。

四种分配算法（必考·选择/简答/大题）

算法	空闲区怎么排	怎么选
① 首次适应 FF	按**地址递增**	从头找，第一个够大的就用
② 循环首次适应 NF	按地址递增	从**上次位置**接着找
③ 最佳适应 BF	按**容量递增**	选**最小的够用**空闲区
④ 最坏适应 WF	按**容量递减**	选**最大的**空闲区

选最适合的最小

选个最大的用

特点：保留高地址大空闲区；低地址碎片多，空间分布均匀；找到大空闲区剩余空间还能用，大作业难满足

示范：空闲区有 100K、500K、200K、300K、600K（按地址顺序），作业依次请求 212K、417K、112K、426K。

首次适应（从头找第一个够的）：从头找，大于它就用，不再往后

- 212K → 找到 500K（第一个≥212的），切走212，剩288K
- 417K → 100K不够、288K不够、200K不够、300K不够、600K够 → 用600K，剩183K
- 112K → 100K不够、288K够 → 用288K，剩176K
- 426K → 100、176、200、300都不够、183不够 → **无法分配**

最佳适应（选最接近的）：在所有大于它中选最小

- 212K → 候选≥212的有300/500/600，选最小的300K，剩88K
- 417K → 候选500/600，选500K，剩83K
- 112K → 候选200/600/100… 选最小够用的200K，剩88K
- 426K → 候选600K，用600K，剩174K → **全部成功**

💡 **结论**：这个例子里最佳适应比首次适应更优。但最佳适应会留下大量小碎片（88K、83K这种）。考试要会按算法一步步切，并说明每种算法的特点。

三、页式存储管理（考纲6）★★★

核心思想

把内存分成固定大小的**页框（块）**，把作业也分成同样大小的**页**。作业的页可以分散放进不连续的页框里，用**页表**记录"哪一页放在哪个页框"。

关键术语

术语	含义
页 / 页面	把作业（逻辑地址空间）分成的等大小块
页框 / 块 / 物理块	把内存（物理空间）分成的等大小块（和页一样大）
页表	记录"页号→页框号"的对应表
页内地址 / 页内偏移	在一页内部的偏移量

页内碎片：作业不一定正好是页大小整数倍，最后一页通常装不满，剩下的空间浪费掉，叫页内碎片

页内碎片（新考纲新增！）★

作业不一定正好是页大小的整数倍，最后一页通常装不满，剩下的空间浪费掉，叫**页内碎片（内部碎片）**。

💡 **页面大小与碎片的关系（新考纲考点）：**

- 页面越大→页表项越少（页表小），但**页内碎片越大**（最后一页浪费多）
- 页面越小→页内碎片越小，但**页表越长**（页太多，页表项多）
- 所以页面大小要**折中**，一般是 2 的幂（如 1KB、4KB）

页式地址转换：

物理地址公式 = 页框号 × 页长 + 页内地址

逻辑地址 → 页号 + 页内地址

=进制

假设页大小 = 2^K

低 K 位 = 页内地址（页内偏移）

高位 = 页号

转换三步骤

逻辑地址：页号 = 逻地址 ÷ 页大小
页内地址 = 逻地址 % 页大小
物地址 = 页框号 × 页大小 + 页内

```
Step 1: 逻辑地址 → 拆出 [页号 P | 页内地址 W]
        页号 P = 逻辑地址 ÷ 页大小（取整）
        页内地址 W = 逻辑地址 mod 页大小

Step 2: 查页表，用页号 P 找到对应的页框号 f
        （顺便检查 P 是否越界：P ≥ 页表长度 → 地址越界中断）

Step 3: 物理地址 = 页框号 f × 页大小 + 页内地址 W
```

f × 页大小 + W

页号P：逻辑地址 ÷ 页大小（取整）
页内地址W = 逻辑地址 mod 页大小
页框号：页号P对应的页框（查页表）
同时检查P之不能大于 → 地址中断

做题示范

例：页大小1KB（1024B），逻辑地址 = 4097，页表显示第4页对应页框号6。求物理地址。

```
Step 1: 页号 = 4097 ÷ 1024 = 4（取整）
        页内地址 = 4097 mod 1024 = 4097 - 4096 = 1
Step 2: 查页表，第4页 → 页框号 6
Step 3: 物理地址 = 6 × 1024 + 1 = 6144 + 1 = 6145
```

二进制做法（更快，考试常用）

如果页大小 = 1KB = 2^10，逻辑地址用二进制表示：

- 低 10 位是页内地址
- 剩下高位是页号

例：逻辑地址 17CAH（十六进制），页大小 1KB

- 17CAH = 0001 0111 1100 1010（二进制）
- 低 10 位（页内地址）= 11 1100 1010
- 高位（页号）= 0001 01 = 5
- 查页表得页框号，替换高位即可得物理地址

这就是你之前真题里 17CAH 那道题的方法。

分页 vs 分段（必考对比）

对比项	分页	分段
划分依据	按固定大小机械划分	按逻辑意义划分（主程序段/子程序段/数据段）
大小	页大小固定	段长度可变
地址	一维（一个逻辑地址）	二维（段号 + 段内地址）
目的	提高内存利用率	便于共享、保护、模块化
碎片	有页内碎片（内部碎片）	有外部碎片
对用户	不可见（系统自动分页）	可见（程序员按逻辑分段）

段页式（结合两者优点）

先分段，每段再分页

逻辑地址 = 段号 + 段内页号 + 页内地址

两张表（每段对应一个页表）+ 页表

优点：既有分段的逻辑清晰，又有分页对内存利用率高

缺点：需查表两次，先查段表，再查页表

页面置换算法

① OPT 最佳置换算法

规则：淘汰未来最长时间内不再被访问的页

缺页率最低，但无法实现（需预知未来）

② FIFO 先进先出

规则：淘汰最先进入内存的页（在内存待最久的）

简单，但可能淘汰常用页，出现 Belady 异常

③ LRU 最近最久未使用

规则：淘汰最近最久没被访问的页（往回看最久没用的）

性能接近 OPT，但实现需硬件支持（记录每页使用时间）

填表规则（三步判断）

每来一个页，问自己三个问题：

```
① 这个页已经在框里了吗？
   是 → 不缺页，三个框原样抄下来，跳到下一个

② 不在框里，那有空框吗？
   有 → 放进空框，记一次缺页✓

③ 不在框里，框也满了？
   → 必须淘汰一个旧页，腾位置给新页，记一次缺页✓
   （淘汰谁，由算法决定：OPT/FIFO/LRU 各不同）
```

OPT：往后看，谁最晚再用就淘汰谁

FIFO：谁先进，淘谁

LRU：往前看，淘汰最久没访问过的页

三、做题方法（核心！画表格）

页面置换题统一画这种表，以课件图的例子演示。

例：页面走向 2 3 2 1 5 2 4 5 3 2 5 2，内存页框数 M=3。

OPT 做法

每次缺页且页框满时，往后面看，淘汰最晚才会再用到的页。

访问	2	3	2	1	5	2	4	5	3	2	5
框1	2	2	2	2	2	2	**4**	4	4	2	2
框2		3	3	3	3	3	3	3	3	3	3
框3				1	**5**	5	5	5	5	5	5
缺页	✓	✓		✓	✓		✓			✓	

注：课件答案显示 OPT 缺页相对少。关键是每次淘汰时往后看哪个最晚用到。

四. 缺页中断次数 / 缺页率

$$\text{缺页率} = \frac{\text{缺页次数}}{\text{总访问次数}}$$

五. Belady 异常

FIFO算法会出现，增加页框数

Day7

一、磁盘调度

磁盘访问时间 = 寻道时间 + 旋转延迟 + 传输时间

- 寻道时间：磁头移动到目标磁道的时间（最主要，磁盘调度就是为了减少它）
- 旋转延迟：目标扇区转到磁头下的时间
- 传输时间：读写数据的时间

> 磁盘调度的目标 = 使平均寻道时间最少（移臂移动距离最短）。

三种调度算法

① 先来先服务 FCFS

按请求到达顺序移动磁头。简单公平，但磁臂可能来回乱跑寻道距离长

② 最短寻道时间优先（SSTF）

每次选离当前磁头最近的请求。性能比先来先服务好，但会饥饿（远处请求一直被插队）

④ SCAN扫描算法（电梯算法）

磁头沿一个方向移动，沿途顺路服务，到头再反向（像电梯：先上到顶，再下来，不会漏人）

做题方法（必考大题）

真题原型：磁盘 200 柱面（0-199），当前磁头在 143，刚完成 125（说明移动方向是从小到大）。请求队列：86，147，91，177，94，150，102，175，130。

三种算法做题结果（真题143柱面）

SSTF 最短寻道：（距磁头最近）

Code

访问顺序：143→147→150→130→102→94→91→86→175→177
总移动 = 4+3+20+28+8+3+5+89+2 = 162

SCAN 电梯（当前方向递增，因为从125→143）：

Code

先往大方向走到最大请求，再掉头：
143→147→150→175→177（最大）→ 掉头 →130→102→94→91→86
总移动 = （177-143）+（177-86）= 34 + 91 = 125

💡 SCAN 算总移动有个快算法：

（最远的同方向请求 - 起点）+（最远的同方向请求 - 最远的反方向请求）

↓

= (177−143) + (177−86) = 34 + 91 = **125**

SCAN-掉头点为最大请求，不一定走到相面边界（199）

除非题目明确"扫到端点"

总移动距离 = 各段距离相加（也叫物理距离/寻道距离）

二、I/O 四种控制方式（查询→中断→DMA→通道）

CPU越来越省心，并行度高

方式	特点	CPU参与度
程序直接查询（轮询）	CPU不断查询设备是否就绪，忙等	最高（全程占用）
中断方式	设备就绪后中断通知CPU，按字节传	较高（每字节中断一次）
DMA方式	成块传送，DMA控制器负责，传完才中断	较低（只在块首尾）
通道方式	独立处理机，CPU只发启动，几乎不参与	最低

三、文件物理结构

文件在磁盘上有三种物理结构

物理结构	怎么存	优点
顺序文件（连续）	占用连续盘块	顺序存取快
链接文件（串联）	盘块离散，用指针链起来	无外部碎片，易扩展，不能随机存取（要顺链）
索引文件	建一张索引表记录每块位置	支持随机存取，易扩展，要索引表占空间

真题考点："逻辑记录可按任意次序存放在不相邻存储块中" → 链接结构

四、混合索引（考纲11）★★★ 最难计算题

你发的那张 i-node 图就是这个。Linux 的 i-node 有 13 个地址项：

```
i.addr[0] ~ i.addr[9]   → 10个直接索引（直接指向数据块）
i.addr[10]              → 1个一次间接索引
i.addr[11]              → 1个二次间接索引
i.addr[12]              → 1个三次间接索引
```

计算最大文件（核心）

设：每个盘块大小 = 1KB，每个地址项 = 4字节 → 一个盘块能放 1024/4 = 256 个地址。

索引类型	能指多少块	说明
直接索引（10个）	10 块	每个 addr 直接指1块
一次间接（1个）	256 块	addr[10]指向一个索引块，里面256个地址
二次间接（1个）	256^2 块	addr[11]→索引块→256个索引块→每个256个地址
三次间接（1个）	256^3 块	再多一层

最大文件块数 = $10 + 256 + 256^2 + 256^3$

混合索引计算结果（必背公式）

```
设：块大小 1KB，地址项 4B → 一块放 256 个地址

最大文件 = 10 + 256 + 256² + 256³ 块
         = 10 + 256 + 65536 + 16777216
         = 16843018 块 ≈ 16 GB
```

习题册原题问法（你真题里那道）：

> 普通文件最大可有 $10 + 256 + 256^2 + 256^3$ 个页

做题套路（三步）

操作... 15 artifacts

Code

```
Step 1: 算"一块放多少地址" = 块大小 ÷ 地址项大小
        （如 1KB ÷ 4B = 256）

Step 2: 各级寻址块数
        直接 = 直接项个数（如10）
        一次间接 = 256
        二次间接 = 256²
        三次间接 = 256³

Step 3: 最大文件块数 = 全部相加
        最大文件大小 = 块数 × 块大小
```

一个块放几个地址。

单独问某级能表示多大文件

- **一次间接**：256 块 × 1KB = **256KB**
- **二次间接**：256^2 块 × 1KB = **64MB**
- **三次间接**：256^3 块 × 1KB = **16GB**

> ⚠️ 注意题目给的**块大小**和地址项大小可能不同（有的块 4KB、地址项4B → 一块放1024地址），公式不变，数字要重