- 第2章 进程与线程
  - **进程 线程**
    - **概念 特征**(2+4
    - **组成**(3
    - **状态转换**(5
    -
    - 控制 (4
    - 通信 (IPC) (3ways
    -
    - 线程(P vs T 多线程模型(3
  - **CPU调度**
    - concept(3级)
    - **实现**
    - **目标** t
    -
    - **进程切换**(first保存→PCB
    - 典型 调度算法(先短 比 片优队
  - **同步 互斥**
    - concept(资源 区;同 斥
    - 临界区互斥的基本方法 (软 硬
    -
    - **互斥锁** (most simple互斥tool
    - 信号量(P V
    -
    - 经典同步问题(P-C R-W Philosopher
    - 管程(封装
  - 死锁
    - concept(4条件
    - **预防**(4
    - **避免 (银行家算法)**(进程Pi
    -
    - **检测**(release **解除**(夺 销
  - 本章疑难点

好的，各位同学，我们继续！第二章《进程与线程》是整个操作系统的**心脏**，也是考研的**重中之重**。这一章概念密集、算法繁多、大题频出。但只要跟上我的节奏，化抽象为形象，再掌握几套解题模板，拿下它不成问题。

第2章 进程与线程

本章我们要搞定四个大魔王：进程与线程、CPU调度、同步与互斥、死锁。每年必考，大题高发区，必须彻底掌握！

进程 线程

概念 特征(2+4

- 进程的定义（与程序的区别）。
- 进程的四大特征：动态性、并发性、独立性、异步性。
- **形象记忆法:**
  - **程序 vs 进程:**
    - **程序 (Program):** 静态的菜谱，写好了放在那里不变。
    - **进程 (Process):** 动态的**做菜过程**，包含了厨师（CPU）、菜谱（代码）、食材（数据）以及做菜的进度（各种状态）。
  - **进程特征 (动并独异):**
    - **动 (动态性):** 进程有生命周期，会创建、运行、消亡。做菜有开始、过程、结束。
    - **并 (并发性):** 多个进程在内存中，宏观上“同时”推进。厨房里可以同时炖着汤、切着菜。
    - **独 (独立性):** 进程是资源分配的**独立单位**。每个大厨有自己独立的厨具和灶台。
    - **异 (异步性):** 进程走走停停，速度不可预知。炖汤的时候你可以去切菜，什么时候切完不确定。

组成(3

- 进程实体由哪三部分组成。
- **形象记忆法:**
  - 进程实体 = 菜谱 (程序段) + 食材 (数据段) + 大脑 (PCB)
  - **PCB (Process Control Block - 进程控制块):** 进程的“身份证”和“灵魂”。
    - **作用:** 操作系统通过PCB来识别、管理和控制进程。

        - **内容:** 包含进程ID、状态、优先级、资源清单、CPU现场（寄存器值）等一切信息。
        - **关键: PCB是进程存在的唯一标志**。创建进程就是创建PCB，撤销进程就是回收PCB。

状态转换(5

- - 五种基本状态：创建、就绪、运行、阻塞、终止。
    - 各状态之间的转换及触发条件（**选择题超高频考点**）。
- **形象记忆法 (赛跑模型):**
    - **创建态:** 运动员正在报名、检录。
    - **就绪..:** 运动员已上跑道，万事俱备，只等裁判（调度程序）发令枪响
    - **运行..:** 枪响后，运动员（**有且仅有一个**）在跑道（CPU）上飞奔
    - **阻塞.. (等待..):** 运动员跑累了，去旁边喝水（等待I/O），**主动放弃**跑道
    - **终止..:** 运动员跑完全程，比赛结束
- **状态转换图 (必背):**
    - **就绪 -> 运行:** 被CPU调度。
    - **运行 -> 就绪:** 时间片用完 / 被更高优先级的进程抢占。
    - **运行 -> 阻塞:** 主动请求I/O或等待某个事件。
    - **阻塞 -> 就绪:** 等待的I/O完成或事件发生。（**注意：不能直接到运行态**，需重新排队等枪响）。

控制 (4

- - 进程控制通过**原语**实现。
    - 创建、终止、阻塞、唤醒 四个过程的主要工作。
- **形象记忆法:**
    - **原语:** 一套不可被中断的指令序列，保证操作的**原子性**。就像银行转账，扣款和存款必须一气呵成。
    - **创建进程:**
        a. 申请一张空白**身份证 (PCB)**。
        b. 分配**资源**（内存、文件等）。
        c. 初始化**身份证**信息。
        d. 插入**就绪队列**排队。
    - **终止进程:**
        a. 找到**身份证 (PCB)**。
        b. 回收**资源**。

c. （如果是父进程）处理所有**子进程**。
d. 销毁**身份证**。

通信 (IPC) (3ways

- 三种高级通信方式：共享存储、消息传递、管道通信。
- **形象记忆法:**
  - **共享存储:** 在两个进程间开辟一块共享的“**小黑板**”。
    - **优点:** 速度快，因为直接读写内存。
    - **缺点:** 需要自己用同步工具（如信号量）来控制，防止写乱。
  - **消息传递:** 进程间通过“**信使**”（内核）来传递格式化的消息。
    - **优点:** 用户不用关心实现细节，方便。
    - **缺点:** 数据需要从用户空间拷贝到内核，再拷贝到另一个用户空间，速度慢。
  - **管道通信:** 一种特殊的共享文件，连接两个进程的“**水管**”。
    - **特点:** 半双工（水只能单向流），先进先出，自带同步互斥。读完就没。

线程(P vs T 多线程模型(3

- 线程与进程的对比（**必考**）。
- 用户级线程与内核级线程的优缺点和区别。
- 多线程模型（多对一、一对一、多对多）。
- **形象记忆法:**
  - **进程 vs 线程:**
    - **进程 (Process):** 一个**工厂**，拥有独立的资源（厂房、设备、原材料）。
      - **线程 (Thread):** 工厂里的**工人**。
    - **对比:**
      - **资源:** 进程是**资源分配**的基本单位（工厂有独立资源），线程**几乎不拥有资源**（工人共享工厂资源）。
      - **调度:** 线程是**独立调度**的基本单位（工人可以独立干活）。

        - **开销:** 创建/切换进程开销大（建新工厂），创建/切换线程开销小（招个新工人/工人换工位）。
        - **并发:** 引入线程后，并发性更高（一个工厂里多个工人可以同时干活）。

- **用户级线程 (ULT) vs 内核级线程 (KLT):**
    - **用户级线程:** “**黑户**”。操作系统（内核）不知道它的存在，线程管理由用户程序自己完成。
        - **优点:** 切换快（不用惊动政府）。
        - **缺点:** 一个线程阻塞（如请求I/O），整个进程（所有线程）都被阻塞。无法利用多核CPU。
    - **内核级线程:** “**有编制**”。操作系统知道每个线程，并由内核来管理和调度。
        - **优点:** 一个线程阻塞不影响其他线程。可以利用多核CPU。
        - **缺点:** 切换慢（需要从用户态到核心态，惊动政府）。
- **多线程模型:**
  ULT → KLT
    - **多对一:** 多个“黑户”线程 对应 一个“有编制”的线程
    - **一对一:** 一个“黑户”线程 对应 一个“有编制”的线程。（Windows, Linux 采用）
    - **多对多:** 多个“黑户”线程 对应 少数几个“有编制”的线程。

CPU调度

concept(3级)

- 三级调度：高级、中级、低级
- **形象记忆法:**
    - **三级调度:**
        - **高级调度 (作业调度):** “**招生办**”。决定把哪些作业从外存（全国考生）调入内存（录取）。控制并发的进程数量。
        - **中.. (内存..):** “**教务处**”。决定把哪个暂时不用的进程从内存（在校生）换到外存（休学生），即“挂起”。

        - **低.. (进程..):** “**任课老师**”。决定就绪队列（班里举手的学生）中下一个谁上CPU（回答问题）。**最基本、最频繁**的调度。

实现

-
    - 调度程序、分派程序
    - 调度时机
    - 调度方式：抢占式、非抢占式
- **形象记忆法:**
    - **调度方式:**
        - **非抢占式:** “**君子模式**”。一旦上了CPU，不主动下（运行完或阻塞）绝不让位。
        - **抢占式:** “**霸道模式**”。更高优先级的来了，或者时间片用完了，就得被强制让位。

目标 t

-
    - CPU利用率、系统吞吐量。
    - 周转时间、等待时间、响应时间。
- **重要公式:**
    - **周转**t = 完成t - 到达t
    - **带权周转**t = 周转t / 运行t
    - **等待**t = 周转t - 运行t (or = 开始执行t - 到达t)

进程切换(first保存→PCB

-
    - 上下文切换的含义和过程。
- **形象记忆法:**
    - **上下文切换:** CPU要换一个进程运行时，必须先**保存当前进程的“工作现场”**（寄存器值等）到它的PCB中，然后再**加载新进程的“工作现场”**。这个过程就像一个演员演完一场戏，卸妆换行头，准备下一场。

典型 调度算法(先短 比 片优队

-
    - 各种调度算法的规则、优缺点。
    - 计算 周转时间、等待时间。（**大题考点**）

- **解题模板 (通用):**
  i. **画图:** 绘制**甘特图**（时间轴），清晰表示每个时刻哪个进程在运行。
  ii. **列表:** 创建一个表格，包含进程名、到达时间、运行时间、完成时间、周转时间、带权周转时间、等待时间。
  iii. **计算:**
    - 根据甘特图，填入每个进程的**完成时间**。
    - 根据公式，计算出**周转时间**和**等待时间**。
    - 计算**平均值**。

- **算法精讲:**
  - **先来先服务 (FCFS):**
    - 按到达时间排队。**公平，但不利于短作业**。
  - **短作业优先 (SJF):** 选择运行时间最短的先跑。
    - **非抢占式:** 选定后跑到完。
    - **抢占式 (SRTN):** 新来了更短的作业，马上换人。
    - **优点:**
      - 平均等待时间、平均周转时间**最优**。
    - **缺点:** 对长作业**饥饿**。

  - **高响应比优先 (HRRN):**
    - $R = \frac{(\text{等待}t + \text{要求服务}t)}{\text{要求服务}t}$
    - **规则:** 每次调度时计算所有就绪进程的响应比，选**R值最大**的。
    - **优点:** 既照顾短作业，又会随着等待时间增加而提高长作业的优先级，**避免饥饿**。

  - **时间片轮转 (RR):** 按FCFS排队，每个进程运行一个**时间片**。用完后回到队尾。
    - **特点:** 公平，响应快，适用于**分时系统**。
  - **优先级调度:**
    - **非抢占式:** 运行中，更高优先级的来了也得等着。
    - **抢占式:** 运行中，更高优先级的来了，立刻抢占。

        - **可能导致饥饿**，可用“老化”技术解决（等待时间越长，优先级越高）。
    - **多级反馈队列:**
      集大成者
        - **设置多个队列**，优先级从高到低，时间片从小到大。
            - 新进程先进最高优先级队列。
            - 在一个队列中时间片用完还没结束，就**降级**到下一队列。
        - 高优先级队列空时，才调度低优先级队列。
        - **优点:** 对各类作业都比较公平，是目前公认较好的算法。

同步 互斥

concept(资源 区;同 斥

-
    - 临界资源、临界区。
    - 同步与互斥的区别。
- **形象记忆法:**
    - 临界资源: 一次只允许一个进程使用的资源，如打印机、共享变量。
    - **临界区:** 访问临界资源的那段**代码**。
    - 同步 vs 互斥:
        - **互斥 (Mutual Exclusion):** 竞争关系。你上厕所时得锁门，别人不能进。是一种**对资源的竞争**。
        - **同步 (Synchronization):** 协作关系。A进程负责生产数据，B进程负责处理数据，必须A生产完，B才能处理。是一种 对执行顺序的**协调**。

临界区互斥的基本方法 (软 硬

-
    - 了解软件（单标志、双标志、Peterson）和硬件（关中断、TSL/Swap指令）实现方法的基本思想和缺陷。
- **记忆要点:**
    - 软件方法
        - 实现复杂且有缺陷（如忙等）。
    - 硬件方法

    - 效率高，但`关中断`对多核无效，`TSL/Swap`仍存在忙等问题。
  - **重点是为引出信号量做铺垫**。

互斥锁 (most simple互斥tool

- - 互斥锁是实现 互斥 的最简单工具。
- **形象记忆法:**
  - 就是一个**二进制信号量**，只有`锁上`和`打开`两种状态。
    - `acquire()`获取锁，`release()`释放锁。

信号量(P V

- - 信号量机制 (PV操作) 的原理与应用（**大题必考**）。
  - 记录型信号量的实现（遵循“让权等待”）。
- **形象记忆法:**
  - **信号量 (Semaphore)** `S`**:** 一个特殊的变量，代表 **资源数量**。
  - **P操作** `wait(S)`**:** 申请资源。`S--`，如果`S < 0`，则进程阻塞等待。
    - **记忆: P**asseren (德语：通过)，想通过先申请。
  - **V操作** `signal(S)`**:** 释放资源。`S++`，如果`S <= 0`，则唤醒一个等待进程。
    - **记忆: V**rijgeven (德语：释放)。
- **PV操作解题模板:**
  - **互斥模型:**
    - 定义互斥信号量 `mutex = 1`
    - 将临界区代码夹在`P(mutex)`和`V(mutex)`之间。

```
P(mutex);       // 锁门
// 临界区代码
V(mutex);       // 开门
```

  - **同步模型:**
    - 定义同步信号量`S = 0`。
    - 在前一个操作的结尾执行`V(S)` (发信号)。
    - 在后一个操作的开头执行`P(S)` (等信号)。

```
// 进程A
...
操作A;
V(S);           // 告诉B，我做完了

// 进程B
...
P(S);           // 等待A完成
操作B;
```

经典同步问题(P-C R-W Philosopher

- - 用PV操作解决三大经典问题是综合应用题的核心。
- **解题模板:**
  - **生产者-消费者问题:**
    - **关系分析:**
      - 生产者与消费者对缓冲区**互斥**。
      - 生产者与消费者之间**同步**（缓冲区满/空）。
    - **信号量设置:**
      - `mutex = 1`: 互斥信号量，用于访问缓冲区。
      - `empty = n`: 同步..，表示空缓冲区的数量。
      - `full = 0`: ..，满..数量
    - **伪代码框架 (必背):**

```
// Producer
P(empty);       // 申请空位
P(mutex);       // 锁
// 放入产品
V(mutex);       // 解锁
V(full);        // 增加产品

// Consumer
P(full);        // 申请产品
P(mutex);       // 锁
// 取出产品
V(mutex);       // 解锁
V(empty);       // 增加空位
```

    - **关键记忆点: 同步P操作在互斥P操作之外**。否则可能导致死锁。

- **读者-写者问题 (读优先):**
  - **关系分析:** 写-写互斥，读-写互斥，**读-读不互斥**。
  - **信号量与变量设置:**
    - `rw = 1`: 保证读写、写写互斥。
    - `mutex = 1`: 保证对计数器 `readcount` 的修改互斥。
    - `readcount = 0`: 记录当前读者数量。
  - **伪代码框架 (核心逻辑):**

```
// Writer
P(rw);
// 写操作
V(rw);

// Reader
P(mutex);
if (readcount == 0) {
    P(rw);          // 第一个读者来，锁门防写者
}
readcount++;
V(mutex);

// 读操作

P(mutex);
readcount--;
if (readcount == 0) {
    V(rw);          // 最后一个读者走，开门让写者
}
V(mutex);
```

- **哲学家进餐问题:**
  - **核心问题:** 5个哲学家，5根筷子，可能发生**死锁**。
  - **死锁原因:** 每个哲学家都拿起左手边的筷子，然后等待右手边的筷子。
  - **解决方案 (任选其一):**
    a. **最多只允许4人同时拿筷子。** (设置信号量 `count=4`)
    b. **同时拿起左右两根筷子。** (将拿筷子操作放入一个 `P(mutex)` 和 `V(mutex)` 之间)。
    c. **奇数号哲学家先拿左后拿右，偶数号反之。**

管程(封装

- - 管程是对信号量机制的封装，是更高级的同步工具。
- **形象记忆法:**
  - **管程 = 带锁的房间**
    - **封装性:** 把共享资源和所有对它的操作都**锁在一个房间里**。
    - **互斥性:** 房间**一次只允许一个人进入**，天然保证了互斥。
    - **条件变量:** 房间里有**等候室 (条件变量)**。如果进入后发现条件不满足（如缓冲区是空的），可以去等候室`wait()`，并**让出房间**给别人用。当别人改变了条件（如放入产品），就去等候室`signal()`一下，唤醒等待的人。

死锁

concept(4条件

- - 定义。
  - 死锁产生的**四个必要条件**（**必背**）。
- **形象记忆法:**
  - **死锁:** 多个进程因互相等待对方持有的资源而造成的僵局。
  - **四个必要条件 (记忆口诀: 互请不循):**
    a. **互 (互斥条件):** 资源是独占的。
    b. **请 (请求和保持条件):** 拿着自己的，还想要别人的。
    c. **不 (不可剥夺条件):** 别人的东西不能抢。
    d. **循 (循环等待条件):** 形成一个“你等我，我等他，他等你”的环路。

预防(4

- - 通过破坏四个必要条件之一来预防死锁。
- **形象记忆法 (破坏疗法):**
  i. 破坏**互斥**: 把独占资源变共享（不现实）。
  ii. 破坏**请求和保持**: 一次性申请所有资源（资源浪费）。
  iii. 破坏**不可剥夺**: 允许抢占资源（实现复杂）。

iv. 破坏**循环等待**: 按序分配资源（限制多）。

避免 (银行家算法)(进程Pi

- - 安全状态的定义。
  - 银行家算法的流程和数据结构。
- **形象记忆法:**
  - **核心思想:** 在分配资源前，先**预判**这次分配会不会导致系统进入“**不安全状态**”。如果会，就不分配。
  - **安全状态:** 系统能找到一个**安全序列** <P1, P2, ..., Pn>，按此序列分配资源，所有进程都能顺利完成。
  - **银行家算法就是找这个安全序列的过程。**
- **解题模板 (银行家算法):**
  i. **准备数据:**
    - **Allocation (已分配)**, **Max (最大需求)** -> 计算 `Need (还需) = Max - Allocation`。
    - Available (可用资源)。
  ii. **安全性检查 (找安全序列):**
    - 设工作向量 `Work = Available`。
    - 从头开始找一个进程 `Pi`，满足 `Need[i] <= Work`。
    - 如果找到，则 `Pi` 可执行完。**假装**它执行完并释放资源：`Work = Work + Allocation[i]`，然后把 `Pi` 加入安全序列。重复此步骤。
    - 如果所有进程都能加入安全序列，则系统**安全**。
  iii. 响应请求:
    - `Pi` 请求 `Request[i]`。
    - **检查1:** `Request[i] <= Need[i]`？(请求是否超额)
    - **检查2:** `Request[i] <= Available`？(资源是否够)
    - 如果都满足，**假装分配**:
      - `Available = Available - Request[i]`
      - `Allocation[i] = Allocation[i] + Request[i]`
      - `Need[i] = Need[i] - Request[i]`
    - 对**假装分配后**的系统状态，**重新运行一遍安全性检查**。
    - 如果安全，则**正式分配**。如果不安全，**撤销假装分配**，让 `Pi` 等待。

检测(release 解除(夺 销

- - 检测：利用简化的资源分配图。
  - 解除：剥夺资源、撤销进程。
- **形象记忆法:**
  - **死锁检测:** “事后诸葛亮”。不预防也不避免，而是定期检查系统里有没有死锁环路。
    - **方法:** 不断寻找可以满足其资源请求并顺利运行结束的进程，然后释放其资源。如果最后还有进程无法被释放，说明它们处于死锁状态。
  - **死锁解除:**
    - **剥夺资源:** 抢！从一个进程手里抢走资源给另一个。
    - **撤销进程:** 杀！强制终止一个或多个死锁进程。简单粗暴。

本章疑难点

- **进程 vs 程序:**
  - 程序是**静态**的指令集合（菜谱），进程是**动态**的执行过程（做菜）。
  - 一个程序可以对应多个进程，一个进程也可以执行多个程序。
- **银行家算法原理:**
  - 核心在于**每次分配前的安全检查**，确保系统总能找到一条出路（安全序列），让所有进程最终都能完成。它是一种**事前避免**策略。
- **同步 vs 互斥:**
  - **互斥**是“**不允许同时**”，是竞争关系。
  - **同步**是“**必须按序**”，是协作关系。
  - 互斥是一种特殊的同步关系。

第二章内容虽多，但脉络清晰。务必掌握进程状态转换、调度算法计算、PV操作解决经典问题和银行家算法这四大核心考点。多动手画图、列表计算，才能真正内化于心。加油！