- 第3章 内存管理
  - 内存管理
    - 原理(3 2address 2redirect 要求(2R
    -
    - **连续分配管理方式**(2区 2片 4Fit
    -
    - 基本分页(PN -P Table->页框 存储管理 2address | TLB'EAT 多level
    - **..段**SN -S Table-> B, L **..** 2..
    - **段页**(S+P**式** 2..
  - 虚拟..
    - concept (3 2
    -
    - **请求分页**(3:PAM | 1中断
    - **页框分配**(2 2 1驻留
    - **页面置换**(4:佳先近钟
    -
    - 抖动 工作集(2
    - **内存映射文件**(2
    - **性能**(缺页率)**影响因素**(4
    -
    - 地址翻译（综合题）(TLB -VP->PP-->Cache

好的，各位同学，我们进入第三章《内存管理》的学习。这一章是操作系统三大核心（进程、内存、文件）之一，也是考研的**高分重灾区**，计算大题频出。内存管理技术的发展史，就是一部想方设法“压榨”有限内存空间的血泪史。跟上我的思路，把物理内存和虚拟内存的来龙去脉彻底搞清楚！

第3章 内存管理

本章分为两大块：**基本内存管理**（物理内存）和**虚拟内存管理**。前者是后者的基础，后者是前者的升华，也是现代操作系统的标配。核心围绕**地址转换**和**空间利用**展开。

内存管理

原理(3 2address 2redirect 要求(2R

- **核心考点:**
  - 程序从代码到执行的完整过程（编译、链接、装入）。
  - 逻辑地址 vs 物理地址。
  - 地址重定位（静态 vs 动态）。
  - 内存保护机制。
- **形象记忆法:**
  - **程序执行三部曲:**
    a. **编译:** 把你写的C++/Java代码（高级语言）翻译成机器认识的**目标模块**（二进制）。
    b. **链接:** 把多个目标模块和你用到的库函数（比如`printf`）“粘”在一起，形成一个完整的**可执行文件**。
    c. **装入:** 把可执行文件从硬盘“搬”到内存里，准备运行。
  - **逻辑地址 vs 物理地址:**
    - **逻辑地址 (相对地址):** 程序自己世界里的地址，比如“变量a在我代码的第100个字节处”。它不关心内存具体情况。
    - **物理.. (绝对..):** 内存条上实实在在的物理位置。
    - **地址转换:** 就像查地图，把“天安门往南1公里” (逻辑地址) 转换成“北京市东城区前门大街甲2号” (物理地址)。这个转换由硬件**MMU (内存管理单元)** 负责。
  - **地址重定位 (搬家):**
    - **静态重定位: 搬家前**就给你定死新家的门牌号。一旦程序装入内存，就不能再动了。
    - **动态..: 搬家后**给你一个动态GPS（**重定位寄存器**）。程序在内存中可以随时“漂移”，每次访问时，CPU都会通过GPS实时计算出物理地址。现代OS都用这种。

  - **内存保护 (私人空间):**
    - **目的:** 防止一个进程去读写另一个进程的内存区域，也防止用户进程破坏操作系统。

        - **实现:** CPU里有两个“门卫”寄存器：
            - **重定位R (基址R):** 记录你家（进程）的起始地址。
            - **界地址.. (限长..):** 记录你家有多大。
            - **访问流程:**
                - CPU每访问一个地址，先检查是否在你家范围内（`逻辑地址 < 界地址寄存器`）
                    - 如果合法，再加上你家的起始地址（`+ 重定位寄存器`），得到最终物理地址。

连续分配管理方式(2区 2片 4Fit

- **核心考点:**
    - 固定分区、动态.. 的特点
    - 动态分区分配的四种算法（首次、最佳、最坏、邻近）。
    - 内部碎片与外部碎片的区别
- **形象记忆法:**
    - **两种碎片:**
        - **内部碎片: 买大了**。给你分配了一块固定大小的空间，你没用完，剩下的就是内部碎片。
            - 比如一个6MB的进程住进8MB的分区，浪费2MB。
        - **外部碎片: 太零碎**。内存里有很多小块的空闲空间，它们加起来很大，但没有一块能容纳下你的进程。
    - **分配方式:**
        - **固定分区:** 内存提前被划成若干**固定大小**的“车位”。车（进程）只能停在大小合适的车位里。会产生**内部碎片**。
        - **动态分区:** 根据来的车（进程）的大小，**现场画一个车位**给它。车走了，车位就擦掉变回空地。会产生**外部碎片**。

    - 动态分区分配算法 (找车位策略):
        - **首次适应 (First-Fit):** 从头开始找，找到**第一个**能停进去的就行。**简单高效，但会留下很多小碎片在内存**

低地址处。

- **最佳.. (Best-..):** 找遍所有空位，找一个 **大小最接近、最“憋屈”** 的停进去。**看似最好，但会产生最多无法利用的小碎片**。

- **最坏适应 (Worst-Fit):** 找遍所有空位，找一个**Max**停进去，剩下的还能给别的车用。**能减少小碎片的产生**。
- **邻近.. (Next-..):** 从上次找到的位置开始找，避免每次都从头扫描。

基本分页(PN -P Table->页框 存储管理 2address | TLB'EAT 多level

- **核心考点:**
  - 页面、页框、页表 的概念
  - 分页系统的 逻辑地址结构
  - **地址变换过程 (计算题核心)**
  - 快表 (TLB) 的作用和有效访问时间的计算
  - 多级页表。

- **形象记忆法:**
  - **分页思想:** 把程序（逻辑空间）和内存（物理空间）都切成**同样大小的块**。
    - 逻辑空间的块叫 **页 (Page)**。
    - 物理..叫 **页框 (Page Frame)** 或物理块。
  - **页表 (Page Table):** 核心！就是一张“**页号 -> 页框号**”的映射表，告诉OS你的第几页放在了内存的第几个框里。

  - **逻辑地址结构:** 逻辑地址 = `页号 P + 页内偏移量 W`。
    - **页号:** 用来**查页表**
    - **页内偏移量:** 在页框内的具体位置，**它在地址转换中保持不变**。

- **template (基本分页地址变换):**
  i. 已知: 逻辑地址 `A`，页面大小 `L` (usually 2^k B)
  ii. **页号 偏移量:**
    - 页号 $P = A/L$ (整除)
    - 页内偏移量 $W = A$ (取余)
  iii. 查找Page Table:
    - 在进程的页表中查找**页号 P** 对应的**页框号 P'**。
  iv. 计算**物理地址:**
    - 物理地址 $E = P' * L + W$

- ..(含快表TLB的**有效访问时间 EAT**):
  - 已知: 快表命中率 `α`，访问快表时间 `t_tlb`，访问内存时间 `t_mem`
  - **核心逻辑:**
    - **命中:** 访问1次快表 + 访问1次内存。
    - **未..:** 访问1次快表 + 访问1次内存(查页表) + 访问1次内存(取数据)
  - 公式 (默认**查页表在内存**):
    - $EAT = \alpha \times (t_{tlb} + t_{mem}) + \underline{(1-\alpha) \times (t_{tlb} + t_{mem} + t_{mem})}$
      - 化简: $= t_{tlb} + \alpha \times t_{mem} + (1-\alpha) \times 2 \times t_{mem}$

- **多级页表** (解决页表过大的问题):
  - **思想:** 给Page table本身也进行分页，建立“页表的页表”，即**页目录**
  - **地址变换:**
    `逻辑地址 = 一级页号(页目录索引) + 二级..(页表..) + 页内偏移量`
  - **访存次数:** N级页表需要访问 N+1 次内存才能得到数据。

..段SN -S Table-> B, L .. 2..

- **核心考点:**
  - 分段的逻辑思想

    - 段表、地址变换过程
    - 分页与分段的对比（**选择题高频**）。

- **形象记忆法:**
    - **分段思想:** 按程序的**逻辑功能**划分，如主函数段、子程序段、数据段、栈段。**段的长度不固定**。
    - **段表 (Segment Table):** “**段号 -> {段基址, 段长}**”的映射表。
    - **逻辑地址结构:** 逻辑地址 = `段号 S + 段内偏移量 W`

- **template (分段地址变换):**
    i. 已知: 逻辑地址 `(S, W)`
    ii. 查找**Segment Table**:
        - 用**段号 S** 去查段表，得到该段的**基址 B** 和**段长 L**。
    iii. 合法性检查:
        - 比较**段内偏移量 W** 和**段长 L**。如果 `W >= L`，则地址越界，产生中断。
    iv. 计算**物理地址:**
        - 物理地址 `E = B + W`

段页(S+P式 2..

- **核心思想:**
    - **分段 分页**的结合体。
        - 先分段，再对每个段进行分页。
- **优点:** 兼具分段（逻辑清晰、易于共享保护）和分页（内存利用率高）的优点。
- **逻辑地址结构:** 逻辑地址 = `段号 S + 段内页号 P + 页内偏移量 W`。

- **地址变换:** 需要**查两次表**（段表 -> 页表）。
    i. 用**段号S**查**段表**，找到该段对应的**页表**的起始地址。
    ii. 用**段内页号P**查**页表**，找到**页框号P'**。
    iii. 物理地址 = `P' * L + W`。

- 一次地址访问需要**3次访存**（段表、页表、数据），通常也用快表来加速。

虚拟..

concept (3 2

- **核心考点:**
  - 为什么需要虚拟内存。
  - 局部性原理（时间局部性、空间局部性）。
  - 虚拟内存三大特征：多次性、对换性、虚拟性。

- **形象记忆法:**
  - **核心思想:** 程序运行时，没必要把所有代码和数据都装入内存，只装入当前要用的就行。
  - **实现:** 利用 **请求分页** 技术，把硬盘当做内存的“扩充”。

  - **局部性原理** (程序的懒人特性):
    - **时间局部性:** 刚用过的东西，马上可能还要用（比如循环里的代码）
    - **空间..:** 刚用了某个地址，接下来很可能要用它旁边的地址（比如数组遍历）
    - 这是虚拟内存**高效运行的理论基础**。

  - **三大特征:**
    - **多次性:** 作业分多次调入内存。
    - **对换..:** 作业的 Page可以在内存和外存之间换来换去
    - **虚拟..:** 从逻辑上扩充了内存容量，让你感觉拥有一个比物理内存大得多的内存空间。

请求分页(3:PAM | 1中断

- **核心考点:**
  - 请求分页的页表机制（增加了哪些标志位）
  - 缺页中断 的处理过程

- **形象记忆法:**
  - **请求分页 页表:** 在基本页表项上增加3个“**开关**”：
    - **状态位 (存在位) P:** 1表示在内存，0..不..
    - **访问.. A:** 最近被访问过置1
    - **修改.. M:** 在内存中被修改过置1（决定换出时是否要写回硬盘）。

  - **缺页中断:**
    a. CPU访问一个逻辑地址，发现页表中对应的状态位 **A=0**。
    b. CPU产生一个“**缺页中断**”异常
      - OS接管，去硬盘找到对应的页
      - **检查内存:**
        - 若有空闲页框，直接装入
        - ..无..，执行**页面置换**算法，淘汰一页
      - 更新页表（修改状态位等）
    c. 返回原指令，重新执行

页框分配(2 2 1驻留

- **核心考点:**
  - 驻留集 的概念
  - 页面分配与置换策略（固定/可变分配，全局/局部置换）。
- **形象记忆法:**
  - **驻留集:** 分配给一个进程的**物理页框**的集合
  - **分配策略:**
    - **固定分配:** 进程运行期间，分到的页框数**不变**。

        - **可变..:** ..，..**可变**..
    - **置换策略:**
        - **局部置换:** 缺页时，只能从**自己**的驻留集里换出一页。
        - **全局..:** ..，可以从**所有**进程的页框中（usually 系统空闲or优先级低的）..

页面置换(4:佳先近钟

- **核心考点:**
    - 四种主要算法的规则和性能。
    - 计算给定页面访问序列下的缺页次数（**大题核心**）。
    - Belady异常

- **解题模板 (通用):**
    i. **画表格:**
        - 第一行是**页面访问序列**。
        - 下面几行代表分配的**物理块 (页框)**。
        - 在表格下方标记每次访问是否**缺页**，以及换出的页面是哪个。
    ii. **模拟过程:** 按照算法规则，一步步填充表格。
    iii. **统计结果:** 统计总的缺页次数。

- 算法精讲:
    - **最佳**置换算法 (OPT): 淘汰**未来最长时间内不会被访问**的页面。
        - **性能最好，但无法实现**，作为衡量其他算法的“标杆”
    - **先进先出 (FIFO):** 淘汰**最先进入内存**的页面。
        - **简单**但**性能差**，可能淘汰常用页。会产生 **Belady异常**
            - （分配的物理块↑，缺页反而↑）

- **最近最久** 未使用 (LRU): 淘汰**最近最长时间**没有被使用的页面。
  - **性能好**，**最接近OPT**，但实现开销大（需要硬件支持来记录访问时间）
- **时钟置换算法 (CLOCK / NRU): LRU**的近似实现，开销小。
  - **规则:** 将所有在内存的页组织成一个环形队列，用一个指针指向下一个要淘汰的候选页。
  - 每个页有一个**访问位A** (初始为1)
  - **淘汰**过程:
    - 检查指针指向的页，看它的访问位A。

    a. `A=1`，给它一次机会，将其置为0
      - 指针下移

    b. .. `=0`，淘汰它，新页换入，访问位置1
      - ..

抖动 工作集(2

- **核心考点:**
  - 抖动（颠簸）的现象和原因
  - 工作集的概念
- **形象记忆法:**
  - **抖动 (Thrashing):** 页面 换入换出过于**频繁**，导致CPU大部分时间都在处理缺页中断，而不是执行程序，系统效率急剧下降。
    - **原因:** 分配给进程的**物理块**太少，连最基本的运行需求都满足不了。
  - **工作集:** 进程在**最近一段时间**（工作集窗口 `Δ`）内 实际访问的**页面集合**

  - **解决抖动:** 操作系统应保证分配给进程的**物理块**数量 **≥** 其**工作集**大小

内存映射文件(2

- **核心考点:**
  - ..的基本思想
- **形象记忆法:**
  - **思想:**
    - 将一个磁盘**文件**“假装”成内存中的一个**数组**
  - **过程:** 程序员可以像访问内存数组一样直接读写文件内容，OS在后台自动处理缺页和数据写回磁盘
  - **优点:** 编程方便，也便于 多进程共享文件（映射同一个文件到各自的地址空间）

性能(缺页率)影响因素(4

- **核心考点:**
  - 影响缺页率的因素。
- **记忆清单:**
  - **页面**大小: 太大内碎片多，太小页表长。
  - **物理块**数量: 分配得越多，缺页越少（但有上限）。
  - 页面**置换**算法: 好算法（如LRU）缺页少。
  - **程序**编写方式: 程序的局部性越好，缺页越少

地址翻译（综合题）(TLB -VP->PP-->Cache

- **核心考点:**
  - 结合TLB、多级页表、Cache的完整地址翻译过程。
- **解题模板 (综合地址翻译):**
  i. 分解**虚拟地址**: 根据页面大小、TLB结构、Cache结构，将虚拟地址分解为 `VPN | VPO` (虚拟页号|页内偏移)
     - 再将VPN分解为 `TLBT | TLBI` (TLB标记|TLB索引)，等等。
  ii. 分解**物理..**: 同理，将物理地址分解为 `PPN | PPO`
     - 再分解为 `CT | CI | CO` (Cache标记|索引||块内偏移)。

---

iii. **模拟访问过程 (三步走):**

- 1 **查TLB**: 用 `TLBI` 和 `TLBT` 查快表
  - **命中 (TLB Hit):** 直接得到物理页号 `PPN`，→ 3
  - **未**.. (.. Miss): -> 2

- 2: 查**Page Table** (慢表): 用 `VPN` (可能分多级) 查内存中的页表
  - **命中** (Page Hit): 得到 `PPN`，将其存入TLB，→ 3
  - **未**.. (.. Fault): 缺页中断，访问结束。

- 3: 查 Cache: 用 `PPN` 和 `PPO` 组合成物理地址，再分解出 `CT` 和 `CI` 查Cache
  - **命中** (Cache Hit): 从Cache中取到数据
  - **未**.. (.. Miss): 从**内存**..，并加载到Cache中

第三章内容技术性非常强，尤其是各种计算。务必亲手多做几道历年真题，把地址变换、EAT计算、页面置换这三大计算模板练到滚瓜烂熟。祝大家学习顺利！