Lecture6 Physical Memroy Management

OS的职责：分配物理内存资源

> - 引入虚拟内存后，物理内存分配主要在以下四个场景出现：
>
> 1. 用户态应用程序触发on-demand paging时
>
>   内核分配物理内存页，映射到对应的虚拟页
> 2. 内核**自己申请内存**并使用时
>
>   内核自身->kmalloc()
> 3. 内核申请用于设备的DMA缓存时
>
>   DMA通常需要连续的物理页
> 4. 发生换页(swapping)时
>
>   通过磁盘扩展物理内存的容量

场景-1：应用触发on-demand paging

- 操作系统需要做(page-fault handler)：
  1. 找到一块空闲的物理内存页（**物理内存管理——页粒度**）
  2. 修改页表，将该物理页映射到触发page-fault的虚地址所在虚拟页
  3. 回到应用，重新执行触发page-fault的代码

物理内存的分配

> 用`alloc_page()`接口实现

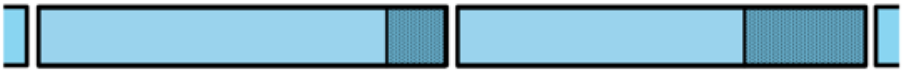
- 操作系统通过**bitmap**来记录物理页是否空闲

物理内存分配器的指标

1. 资源利用率 2. 分配性能

– 外部碎片与内部碎片

外部碎片：单个空白部分都小于分配请求的内存大小，但加起来足够
注：蓝色部分表示已分配内存，空白部分为未分配内存



内部碎片：蓝色阴影部分是分配内存大于实际使用内存而导致的内部碎片
注：黑色粗线框表示已分配内存，蓝色部分表示实际使用内存，蓝色阴影表示已分配但未被使用部分

伙伴系统(buddy system)

> 类似我们在ICS中实现的malloc lab

描述物理页的数据结构

```
struct physical_page {
  // 是否已经分配
  int allocated;
  // 所属伙伴块大小的幂次
  int order;
  // 用于维护空闲链表，把该页放入/移出空闲链表时使用
  list_node node;
};

// 伙伴系统的空闲链表数组
list free_lists[BUDDY_MAX_ORDER];
```

操作系统维护struct physical_page数组

> Buddy System是一个管理物理页分配的数据结构，存放在内核中，映射到物理内存，上图是它的数据结构↑
>
> Kernel也可能发生Page Fault（e.g.传递syscall的参数），但是能超过**3次**，否则会***Panic***
>
> 对于高地址(kernel)，map了但不一定use；对于低地址(user), map了一定use，use了不一定map（因为可能一开始没分配物理地址）

细粒度内存管理

场景-2：内核运行中需要进行动态内存分配

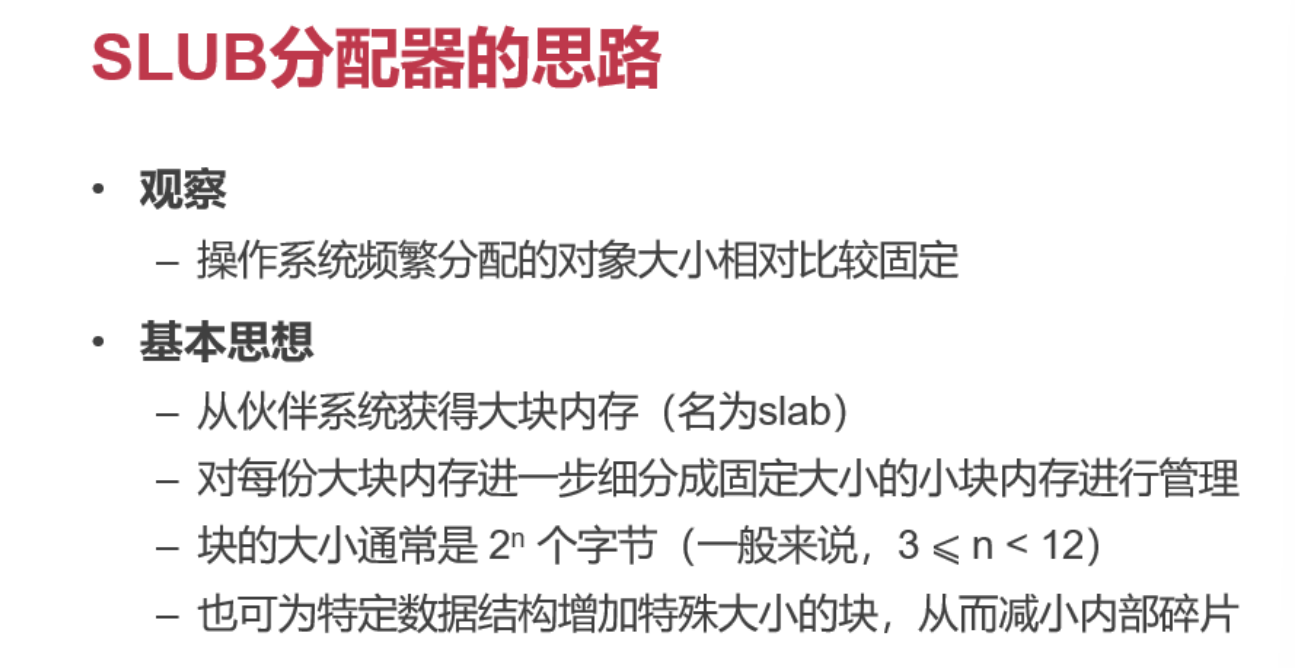
- 内核自身用到的数据结构
    1. 为每个进程创建的process，VMA等数据结构
    2. 动态性：用时分配，用完释放，类似于用户态的malloc
    3. 数据结构大小往往小于页粒度

SLAB：建立在伙伴系统之上的分配器

> 目标：快速分配小内存对象（内核中的数据结构大小远远小于4K）

- SLAB分配器家族：SLAB，SLUB，SLOB

SLUB分配器



> 可以理解为A>U，所以slab的是大块内存

- SLUB的设计
    1. **只分配固定大小块**
    2. 对于每个固定块大小，SLUB分配器都会使用独立的内存资源池进行分配
    3. 采用***best fit***定位资源池

        > 从全部空闲内存块中找出能满足要求且大小最小的空闲内存块

SLUB小结

优势：

1. 减少内部碎片（可根据开发需求）
2. 分配效率高（常数时间）

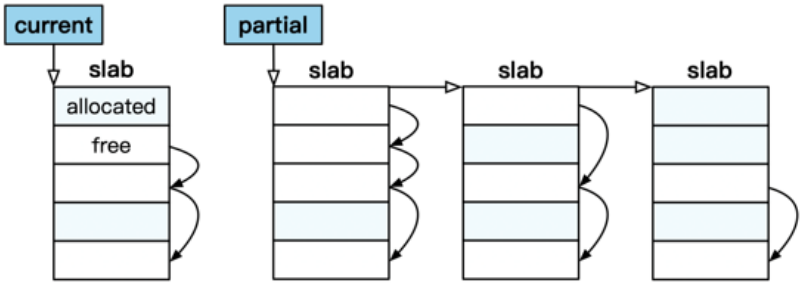
针对每种slot大小维护两个指针：

- current仅指向一个 slab
  - 分配时使用、按需更新
- partial指向未满slab链表
  - 释放时若全free，则还给伙伴系统

从伙伴系统获得的物理内存块称为 slab

slab内部组织为空闲链表

1. 思考：选择哪些slot大小?
2. 思考：分配与释放的时间复杂度?



> A1：二的幂次，最常用的
>
> A2：O(1)，直接用next_free指针指向的slot就好

突破物理内存容量限制

> 场景3：物理内存容量 < 应用进程需求

换页机制(Swapping)

- **换页的基本思想**
  - 用磁盘作为物理内存的补充，且对上层应用透明
  - 应用对虚拟内存的使用，不受物理内存大小限制
- **如何实现**
  - 磁盘上划分专门的Swap分区，或专门的Swap文件
  - 在处理缺页异常时，触发物理内存页的换入换出

问题1：如何判断缺页异常是由于换页引起的

- 导致缺页异常的三种可能：

1. 访问非法地址
2. 按需分配（尚未分配真正的物理页）
3. 内存页数据被换出到磁盘上

- OS如何区分？
    1. 利用VMA区分是否为合法虚拟地址（合法缺页异常）
    2. 利用页表项内容区分是否按需分配还是需要换入

> 在PTE中存放***swap bit***，在page fault后在内核代码查看。若是swap出去的，则读取PTE的物理地址（此时存放的是disk中该page的位置）访问disk；若否则需要按需分配（说明还未分配真正的物理页）
>
> > 此时的Valid bit一定是0！！！

问题2：何时进行换出操作

- **策略A**
  - 当用完所有物理页后，再按需换出
  - 回顾：alloc_page，通过伙伴系统进行内存分配
  - 问题：当内存资源紧张时，大部分物理页分配操作都需要触发换出，造成分配时延高
- **策略B**
  - 设立阈值，在空闲的物理页数量低于阈值时，操作系统择机（如系统空闲时） 换出部分页，直到空闲页数量超过阈值
  - Linux Watermark：高水位线、低水位线、 最小水位线

回顾：延迟映射 vs. 立即映射

- **优势：节约内存资源**
- **劣势：缺页异常导致访问延迟增加（换页面临相似问题）**
- **如何取得平衡?**
  - 应用程序访存具有时空局部性（Locality）
  - 在缺页异常处理函数中采用预先映射的策略
    - 即节约内存又能减少缺页异常次数

> Prefetching：预先多加载几个页（利用了locality）

问题3：换页机制的代价

- **优势：突破物理内存容量限制**
- **劣势：缺页异常+磁盘操作导致访问延迟增加**
- **如何取得平衡?**
  - 预取机制 （Prefetching）
    - 预测接卸来进程要使用的页，提前换入
    - 在缺页异常处理函数中，根据应用程序访存具有的空间本地性进行预取

问题4：如何选择换出的页

> - 页替换策略
>   1. 选择一些物理页换出到磁盘
>   2. 猜测哪些页面**应该**被换出（短期内大概率不会被访问）
>   3. 策略实现的开销

- 理想的换页策略(OPT策略)

> OPT：优先换出未来最长时间内不会再访问的页面

> （有点像先知）

- FIFO策略

> OS维护一个队列用于记录换入内存的物理页号
>
> 最先进入的最先换出（就是队列）

- Second Chance策略

> FIFO策略的一种改进版本，解决***Belady's Anomaly***

FIFO 策略的一种改进版本：为每一个物理页号维护一个访问标志位。

如果访问的页面号已经处在队列中，则置上其访问标志位。

换页时查看队头：1）无标志则换出；2）有标志则去除并放入队尾，继续寻找

物理页面访问顺序	3	2	3	1	4	3	5	4	2	3	4	3
该行是队列头部	3	3	3*	3*	1	1	3*	3*	3	3*	3*	3*
FIFO 队列		2	2	2	3	3*	4	4*	4	4	4*	4*
存储物理页号				1	4	4	5	5	2	2	2	2
缺页异常（共 6 次）	是	是	否	是	是	否	是	否	是	否	否	否

- LRU策略

OS维护一个链表，在每次内存访问后，OS把刚刚访问的内存页调整到链表尾端；每次都选择换出位于链表头部的页面

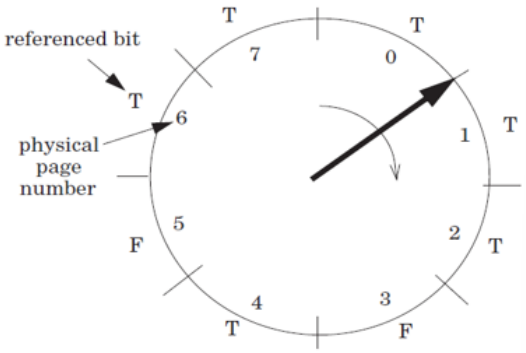
缺点-1：对于特定的序列，效果可能非常差，如循环访问内存

缺点-2：需要排序的内存页可能非常多，导致很高的额外负载

物理页面访问顺序	3	2	3	1	4	3	5	4	2	3	4	3
该行为链表头部	3	3	2	2	3	1	4	3	5	4	2	2
越不常访问的页号		2	3	3	1	4	3	5	4	2	3	4
离头部更近				1	4	3	5	4	2	3	4	3
缺页异常（共 7 次）	是	是	否	是	是	否	是	否	是	是	否	否

- 时钟算法策略

时钟算法策略



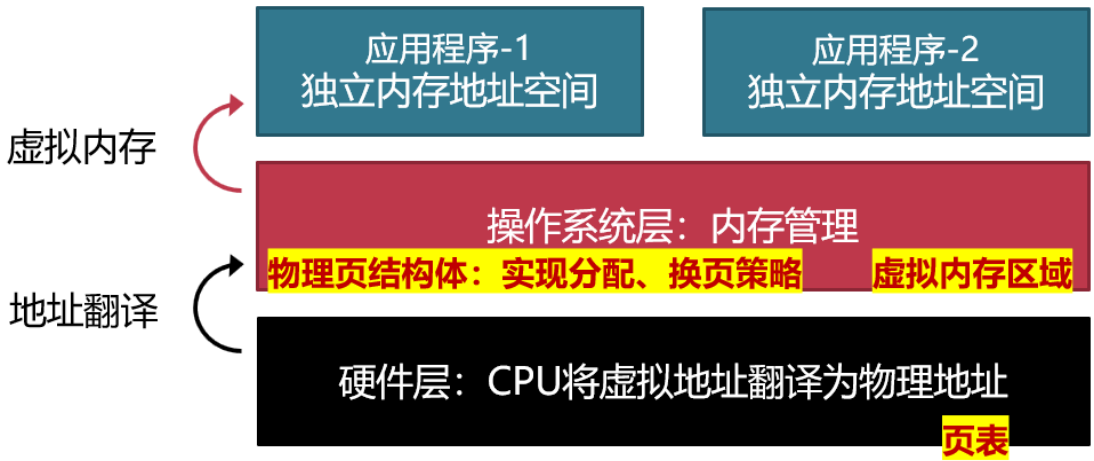
- 精准的LRU策略难以实现
- 物理页环形排列（类似时钟）
  - 为每个物理页维护一个访问位
  - 当物理页被访问时， 把访问位设成T
  - OS依次（如顺时针）查看每个页的“访问位”
    - 如果是T，则置成F
    - 如果是F，则驱逐该页

> 换入时写的是虚拟地址，先写通过kernel中虚拟地址映射的物理地址，然后再修改user space中的页表项

- 小结

  - **常见的替换策略**
    - FIFO、LRU/MRU、时钟算法、随机替换 …
  - **替换策略评价标准**
    - 缺页发生的概率 （参照理想但不能实现的**OPT策略**）
    - 策略本身的性能开销
      - 如何高效地记录物理页的使用情况?
        - 页表项中Access/Dirty Bits
  - **小知识：颠簸现象 Thrashing Problem**

Summary


应用程序-1
独立内存地址空间
应用程序-2
独立内存地址空间
虚拟内存
操作系统层：内存管理
物理页结构体：实现分配、换页策略
虚拟内存区域
地址翻译
硬件层：CPU将虚拟地址翻译为物理地址
页表