Tiny Buddy Book

A simple implementation of Buddy memory allocation

ucore学习笔记 Author: Ding Boyang

The physical memory abstracion

We divided whole memory space into many chunks with the same size, and define each chunk as 4KB in conventional ,which is the size of a page in x86-32 platform. Then in uCore we defien data type `struct Page` and use an array of `pages` to represent entire physical memory, our physical memory manager(**pmm**) is built based on this array.Another thing we should notice is that the usable physical memory is not the continuous address space in ideal, for example ,some memory is reserved for kernel,or perhaps memory chipset on motherboard lacks some chips, which lead to “memory hole” . we flag these unallocatable pages so that they will be never touched by any processes. The pages that are not flaged make up the free area.The responsibility of **pmm** is allocate allocatable pages from free area to process and reclaim freed memory to free area.

Buddy system as a pmm for uCore

Based on the page structure we have described above, we use a powerful tool to manage physical memory.
The traditional pmm just like **first-fit**,**best-fit**, etc. neraly all use doublely linked list to manage free pages,this cause the allocation or free algorithm is $O(n)$ in complexity at worst,if the page number is very large ,this seems too slow.
Buddy memory allocation describe a series exquisite algorithms that can finish allocation and free work at worst $O(logn)$
Now let’us treat the memory as a complete binary tree(CBT). Sounds a little stranger,right ? First, walk into the bottom of this CBT,we can see many tree node stand in a line,define every node as a page and are continuous in address. So these page make up entire memory space, and then go up stairs, every higher level nodes can be treated as a root of subtree, this subtree owns some pages in his bottom, and this is a subset of memory space,the higher level it is, the more pages it owns, the root of CBT owns whole memory space.Thus we treat each node as a memory block and each block is exactly power of 2 pages in size.The reason why we use CBT to represent memory map is that we can use binary search in allocation algorithm, it’ll bring $O(logn)$ in complexity.

- **Buddy Data Structure**
  It is time to define the datastrcture of this CBT，before to do this , we need to make clear how to measure size of memory. Block size are be measured using term `order`, a block with order `o` means this block have 2^{o-2} continuous pages, the start address of this block is start address of first block. Every node we called it a buddy, a buddy is just a unsigned short which separate into three field as follows.

order	left	right

  - order : The order of this block.
  - left : The allocable memory of left tree, measure in order
  - right : The allocable memory of right tree, measure in order

  Each field is 5 bit width, the most significant bit is not used at present. We can extract this info through bit-wise operation.
  The CBT is an array of buddys, which is defined as `Buddy *mem` in uCore .We need a continuous memory reserved for Buddy system.
- **The allocation algorithm**
  The algorithm is just like binary search. Given the number of pages need to allocated , calculate the minimum order that can satisfy requirement.Then start from root,if the left allocable satisfy request then go left,else if the right allocable satisfy request then go right, if neither left nor right could satisfy request, then the allocation failed and return `NULL`,otherwise it will finally stop at a buddy. and this buddy's order must equal allocation order, it is what we want. Attention! Don't forget flag this buddy as unallocatable and update its ancestor's allocable status. `struct Page* buddy_alloc_pages(size_t n)` do this work in uCore, return value is the head page of a memory block, and the page's property flag should be set, the property should be assigned to n, which means the next following continous pages are allocated.
- **The free algorithm**
  Given an allocated block,first we should translate physical address to buddy index,then modify buddy's field. Next flag this buddy as allocatable and update ancestors' allocatable status. Because of using of CBT,free block merge has finished implicitly. `void buddy_free_pages(struct Page* page)` do this work in uCore, the argument page is the header page of memory block need to be freed, the page's property flag should be clean and property should be assigned to zero.
- **The initialization algorithm**
  Initialization is divided into 2 steps.

  - Flag all pages as unallocable
  - Flag usable page as allocable one by one
    Bootloader will provide usable continous pages to pmm, once we flag one page as allocable, update his ancestor's allocable status immediately. Each page will cost $O(logn)$ to do update, The complexity of this algorithm is finally

$O(nlogn)$

`static void buddy_init_memmap(struct Page* page, size_t n)` do this work in uCore. If `mem_init` equals false, which means `Buddy* mem` has not been initialized, let `mem` point to a continous memory block with size of CBT in the front of detected memory, so these memory are be reserved for buddy system. then call `buddy_fill(0)` to flag all pages as unallocable.

- **The update algorithm**
  We have mentiond update many times. Now let's see this important algorithm who guarantee a buddy's allocable info is always right.Each buddy has its left tree and right tree, we can set buddy's left/right field as allocable(left tree/right tree). what is allocable size of a buddy? If this block is totally free, which means no pages is allocated in this block, the allocable size is just the size of this buddy. else the allocable size is maximum of allocable(left tree) and allocable(right tree) . When an allocation or a free happend in a buddy, the allocable size of his ancestor might be changed, update ancestor's left and right field iteratively until reach root. `static void update(int i)` do this work in uCore.