Buddy System内存管理算法

引言

Buddy System 是一种用于管理物理内存的算法，它将可用的物理内存划分为不同大小的块，并以二叉树的形式组织。该算法可以高效地分配和释放内存，同时有效地解决了内存碎片化的问题。本文档旨在介绍 Buddy System 的设计原理、数据结构以及操作流程。

设计原理

- 将整个物理内存划分为大小为2的幂次方的块，每个块都有一个对应的二叉树节点。
- 如果一个块被分配，它会被进一步划分为两个较小的块。相邻的两个块合并时，它们将形成一个更大的块。
- 使用一个空闲链表来跟踪每个块的可用状态，空闲链表按照块的大小递增排列。
- 对于参数n的处理：如果n是2的整数次幂，要分配/释放的页数直接取n，否则取大于n的最小2的整数次幂。
    - 使用函数：

```
static unsigned fixsize(unsigned size) {
    size |= size >> 1;
    size |= size >> 2;
    size |= size >> 4;
    size |= size >> 8;
    size |= size >> 16;
    return size+1;
}
```

数据结构

- buddy2结构体：二叉树节点
    - size：表明二叉树管理总内存的大小
    - longest：该节点所能分配的最大内存
- allocRecord结构体：记录分配块的信息
    - Page* base：分配块的起始地址
    - offset：偏移量
    - nr：块大小
- 结构体实例及其他变量：
    - root[80000]：存放二叉树的数组
    - rec[80000]：存放分配块信息的数组
    - nr_block：已分配块数
    - nr_free：空白页数
    - free_list：管理页的双向链表

初始化二叉树节点

先判断传入的节点size是否满足条件，之后给根节点的size属性赋值，其他节点的longest属性按照对应的2降幂赋值即可。

```
for (i = 0; i < 2 * size - 1; ++i) {
    if (IS_POWER_OF_2(i+1))
      node_size /= 2;
    root[i].longest = node_size;
  }
```

初始化内存映射关系

该初始化函数与best_fit的初始化内存函数较为类似，初始化Page各种状态并将其加入空闲链表，通过传入的参数n计算所要初始化的二叉树节点数，然后调用初始化二叉树节点的函数，完成映射。

```
int allocpages=UINT32_ROUND_DOWN(n);
buddy2_new(allocpages);
```

内存分配

- int buddy2_alloc(struct buddy2* self, int size);

首先对能否分配进行判断，返回相应值；如果可进行分配，则从二叉树根节点开始循环遍历左右子树，找到两个相符合的节点中内存较小的结点，最后的index即为要分配的块在二叉树中的索引。

```
for(node_size = self->size; node_size != size; node_size /= 2 ) {
    if (self[LEFT_LEAF(index)].longest >= size){
       if(self[RIGHT_LEAF(index)].longest>=size){
           index=self[LEFT_LEAF(index)].longest <=
self[RIGHT_LEAF(index)].longest? LEFT_LEAF(index):RIGHT_LEAF(index);
         //找到两个相符合的节点中内存较小的结点
        }
       else{
         index=LEFT_LEAF(index);
       }
    }
    else
      index = RIGHT_LEAF(index);
  }
```

之后将该节点的longest置0表示节点已被使用，再根据index求出该块在整个内存空间中的偏移量：

```
offset = (index + 1) * node_size - self->size;
```

最后由于修改了二叉树的某一节点，所以从该节点向上递归，修改父节点的longest值为MAX{左孩子longest，右孩子longest}，返回值为offset。 根据该节点我们可以得到要分配的起始页在内存中的偏移量。

- static struct Page* buddy_alloc_pages(size_t n)

判断n为可分配页面后调用buddy2_alloc()获取偏移量并记录到本次分配对应的rec数组中。之后根据偏移量在空页面链表中取出对应的页面指针从而获取该页结构体，将该起始页和计算出的分配页数存入rec数组。下一步设置已分配的页状态和更改nr_free参数值，最后返回page，来表示分配块的基址，这样就完成了对块的分配。

内存释放

- void buddy_free_pages(struct Page* base, size_t n)

遍历rec数组，找到与传入base相同的块基址，从而获取该块对应的偏移量，根据偏移量计算出该页在二叉树中对应的索引：

```
index = offset + self->size - 1;
```

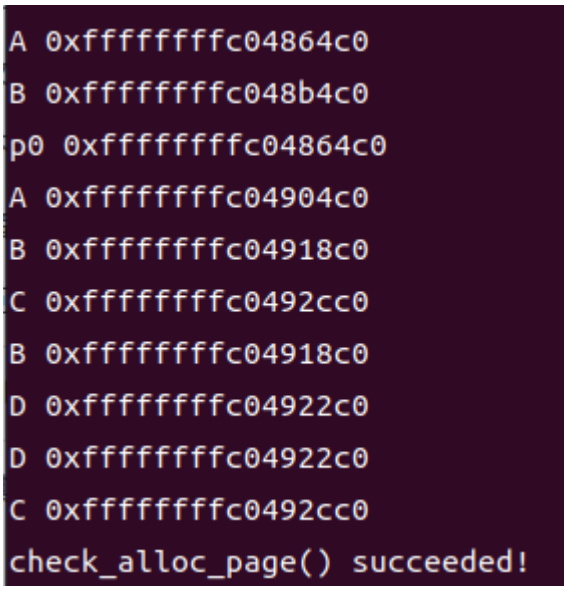
对于我们的二叉树数组，由于将某一个节点的longest值修改为allocpages，因此要向上递归，保证父节点可分配页数的正确性，如果左右子树longest相加等于该节点初始对应的node_size，代表这又是一个可连续分配的内存区间，否则就取最大值。

```
while (index) {//向上合并，修改先祖节点的记录值
    index = PARENT(index);
    node_size *= 2;
    left_longest = self[LEFT_LEAF(index)].longest;
    right_longest = self[RIGHT_LEAF(index)].longest;
    if (left_longest + right_longest == node_size)
      self[index].longest = node_size;
    else
      self[index].longest = MAX(left_longest, right_longest);
  }
```

然后对其他比如分配记录和空闲页数进行修改，并回收已分配的页。

check部分

我们调用上面的分配和释放函数进行内存块的申请和释放，并使用assert检查是否满足buddy system的分配情况，结果如下：



说明Buddy System编写成功!