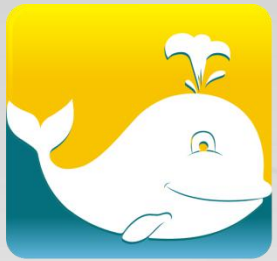


操作系统

Operating System

伙伴系统(Buddy System)

- 整个可分配的分区大小2^U
- 需要的分区大小为$2^{U-1} < s \leq 2^U$时，把整个块分配给该进程；
  - 如$s \leq 2^{i-1}$，将大小为2^i的当前空闲分区划分成两个大小为2^{i-1}的空闲分区
  - 重复划分过程，直到$2^{i-1} < s \leq 2^i$，并把一个空闲分区分配给该进程

伙伴系统的实现

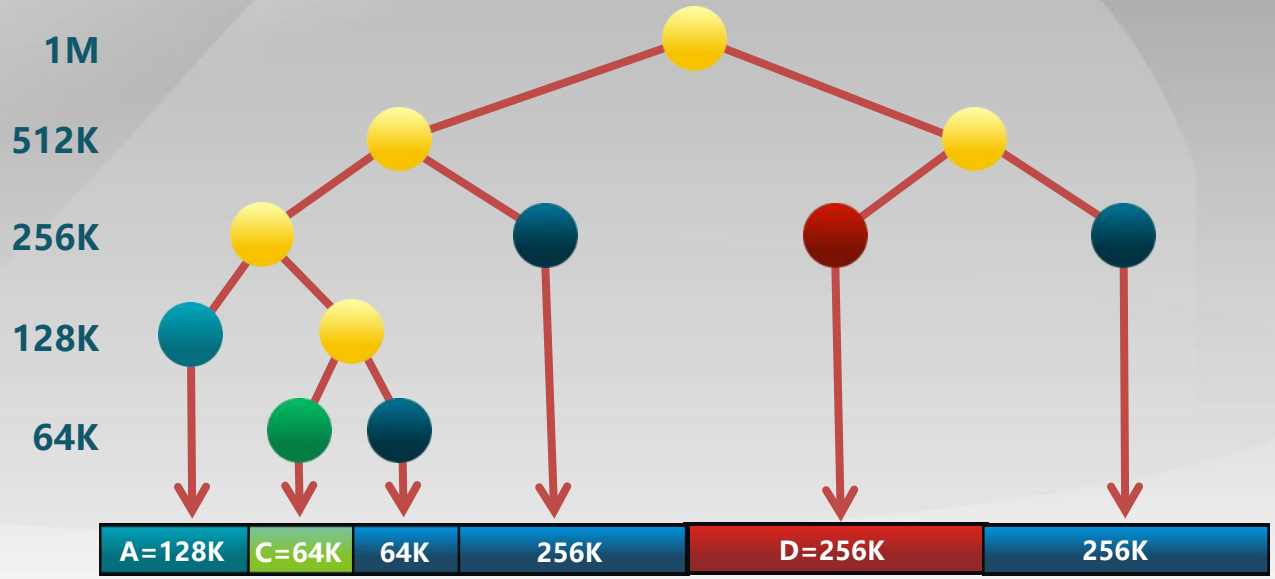
- 数据结构
  - 空闲块按大小和起始地址组织成二维数组
  - 初始状态：只有一个大小为2^U的空闲块
- 分配过程
  - 由小到大在空闲块数组中找最小的可用空闲块
  - 如空闲块过大，对可用空闲块进行二等分，直到得到合适的可用空闲块

伙伴系统中的内存分配


1 Mbyte block
1M
Request 100K
A=128K
128K
256K
512K
Request 240K
A=128K
128K
B=256K
512K
Request 64K
A=128K
C=64K
64K
B=256K
512K
Request 256K
A=128K
C=64K
64K
B=256K
D=256K
256K
Release B
A=128K
C=64K
64K
256K
D=256K
256K
Release A
128K
C=64K
64K
256K
D=256K
256K
Request 75K
E=128K
C=64K
64K
256K
D=256K
256K
Release C
E=128K
128K
256K
D=256K
256K
Release E
512K
D=256K
256K
Release D
1M

伙伴系统的实现

- **释放过程**
  - 把释放的块放入空闲块数组
  - 合并满足合并条件的空闲块
- **合并条件**
  - 大小相同2^i
  - 地址相邻
  - 低地址空闲块起始地址为2^{i+1}的位数



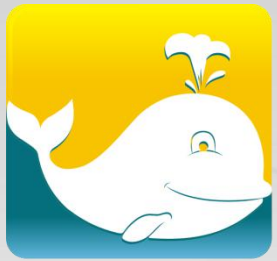
http://en.wikipedia.org/wiki/Buddy_memory_allocation

ucore中的物理内存管理

```
struct pmm_manager {
    const char *name;
    void (*init)(void);
    void (*init_memmap)(struct Page *base, size_t n);
    struct Page *(*alloc_pages)(size_t order);
    void (*free_pages)(struct Page *base, size_t n);
    size_t (*nr_free_pages)(void);
    void (*check)(void);
};
```

ucore中的伙伴系统实现

```
const struct pmm_manager buddy_pmm_manager = {
    .name = "buddy_pmm_manager",
    .init = buddy_init,
    .init_memmap = buddy_init_memmap,
    .alloc_pages = buddy_alloc_pages,
    .free_pages = buddy_free_pages,
    .nr_free_pages = buddy_nr_free_pages,
    .check = buddy_check,
};
```



操作系统

Operating System