实验 6 伙伴算法的实现

目的：理解伙伴系统的工作原理，设计算法实现基于伙伴系统的内存管理算法。

原理：固定和可变分区内存管理方法都存在缺陷。固定分区由于分区数目是固定的，因此限制了活动进程的个数，而且当可用内存大小与进程内存需求大小不匹配时，内存使用效率非常低效。而可变分区方法管理起来较为复杂，而且由于进程镜像的上下浮动会带来额外开销。伙伴系统是二者的一种折中，兼具固定和可变分区的优点。

在伙伴系统中，内存块个数和大小是不固定的，但是大小只能限定为 2^K 个字节，其中

$$L \le K \le U$$

- 2^L = 最小块的大小（字节）
- 2^U = 最大块的大小（字节）。通常是整个内存的大小。

开始时，整个可供内存分配的空间被看做一个块，其大小为 2^U。如果请求的内存大小为 s，且 $2^{U-1} < s \le 2^U$，那么把整个内存，即 2^U 分配给它。否则将整个内存空间分为两个大小为 2^{U-1} 的伙伴块，并将其中的一个暂时分配给该请求。然后再考查，该 2^{U-1} 大小的块中，条件 $2^{U-2} < s \le 2^{U-1}$ 是否满足，若满足，则将该 2^{U-1} 大小的块正式分配给该请求，然后退出内存分配算法；若不满足，则将该 2^{U-1} 大小的块继续一份为二，产生两个大小为 2^{U-2} 的伙伴块，并把其中一个暂时分配给该请求。这样的过程持续下去，直到如下条件满足时，内存分配算法退出：

（1） 存在 $L < i \le U$，使得 $2^{i-1} < s \le 2^i$ 时，将大小为 2^i 的块分配给该请求，算法退出；

（2） $s \le 2^L$ 时，把大小为 2^L 的块分配给该请求，算法退出。

要求：

1、 采用自己熟悉的语言编写，如 C/C++/Java

2、 使用二叉树数据结构管理和维护内存中的若干分区状态

二叉树的一个叶子节点代表一个分区，非叶子节点的两个子女节点分别代表一个分区被分为两个大小相等的伙伴。

树节点的数据结构为：

```
struct chunk{
    unsigned int ID;          //分区编号
    int status;               //分区状态，1 表示已分配，0 表示空闲
    int size;                 //分区大小，只能为 2^L ~ 2^U
    struct chunk* leftBuddy;  //左伙伴
    struct chunk* rightBuddy; //右伙伴
}
```

3、 在二叉树上定义两个操作：

- int Allocate(int size)

  说明：请求分配一个 size 大小的内存

  返回值：若返回值大于等于 0，则表示分配成功，返回值为分区的编号；若返回值

<0，则表示分配失败。

注意：在 Allocate 执行过程中可能伴随内存块的折半划分，因此二叉树结构有可能发生变化。

- int Release(unsigned int ID)

  说明：释放编号为 ID 的内存块

  返回值：0：表示释放成功，<0 表示释放失败。

  注意：在 Release 执行过程中，可能伴随 Buddy 合并，使得二叉树的结构发生变化。当一个区块的 leftBuddy 和 rightBuddy 都为空闲状态时，两个 Buddy 合并为一个空闲 Buddy。

- int printMemMap( )

  说明：打印出当前所有内存区块（按从左至右的顺序）的编号、状态和大小情况。

  返回值：0：表示正确打印，<0 表示打印失败。

4、算法说明

- 开始时内存大小为 $M=2^U$，且处于未分配状态。
- 随着 Allocate 和 Release 的不断执行，二叉树结构不断发生变化，树结构的变化反映了内存区块的状态变化。

5、算例

令 M=1M 字节，即 U=20。L=10，即最小块为 1KB。

执行如下内存管理操作：

```
printMemMap( );
ID1=Allocate(100K);        //如果成功分配，则将返回的内存编号赋值给 ID1
printMemMap( );
ID2=Allocate(240K);
printMemMap( );
ID3=Allocate(64K);
printMemMap( );
ID4=Allocate(256K);
printMemMap( );
Release(ID2);
printMemMap( );
Release(ID1);
printMemMap( );
Allocate(75K);
printMemMap( );
Release(ID3);
printMemMap( );
Release(ID4);
printMemMap( );
```

给出上述序列中，每条 printMemMap()的执行结果。