伙伴算法：动态内存管理的高效方法

指导教师：杨刚

2023 年 11 月 13 日

伙伴算法（Buddy Algorithm）通常用于管理动态分配的内存块，特别是在操作系统内存管理和文件系统等领域。伙伴算法的主要应用的场景有：

- 内存管理

  动态内存分配 伙伴算法可用于动态分配内存块，其中内存被分割成块，每个块的大小是 2 的幂。这种分割使得对内存的分配和释放更加高效。

  碎片管理 伙伴算法有助于减少内存碎片，通过在释放内存时将相邻的空闲块合并为较大的块，从而提高内存利用率。

- 文件系统

  磁盘空间分配 在文件系统中，伙伴算法可以用于磁盘空间的分配和释放。文件系统通常将磁盘空间分割成块，使用伙伴算法来管理这些块。

  碎片整理 类似于内存管理，伙伴算法在文件系统中也有助于减少碎片，提高磁盘空间的利用率。

- 缓存管理

  缓存块分配 在缓存系统中，伙伴算法可以用于分配和释放缓存块，以提高对数据的访问效率。

- 通信系统

  消息缓冲区管理 在通信系统中，伙伴算法可以用于管理消息缓冲区，以优化消息的接收和发送。

总体而言，伙伴算法在需要进行动态分配和释放资源的场合非常有用，尤其是在需要处理块状资源的情况下，它可以有效地管理资源并提高系统性能。

请基于下面这个伙伴算法的不完整模拟实现，完成：

问题 1.

认真阅读并理解这个程序，并给出其执行结果[1]。

问题 2.

请通过`reclaim(std::pair<int, int> range)`完善该实现中未完成的已分配内存块的回收。

Be active!

小组完成。截止期为 +2 周。

[1]在分析的基础上给出结果，请勿通过执行获得答案。

```cpp
#include <vector>
#include <map>
#include <cmath>
#include <iostream>
using namespace std;

void initialize(int sz);
void allocate(int sz);

// Size of vector of pairs
int length=0;

// Global vector of pairs to store address ranges available in free list
vector<pair<int, int>> free_list[100000];

// Map used as hash map to store the starting address as key
// and size of allocated segment key as value
map<int, int> mp;

int main()
{
  initialize(128);
  allocate(32);
  allocate(7);
  allocate(2);
  allocate(64);
  allocate(56);

  return 0;
}
```

Listing 1: 主程序

```
32
33 void initialize(int sz)
34 {
35   // Maximum number of powers of 2 possible
36   int n = static_cast<int>(ceil(log(sz) / log(2)));
37   length = n + 1;
38   for (int i = 0; i <= n; i++) free_list[i].clear();
39   // Initially whole block of specified size is available
40   free_list[n].push_back(make_pair(0, sz - 1));
41 }
```

Listing 2: 数据初始化

```
43 void allocate(int sz)
44 {
45   // Calculate index in free list to search for block if available
46   int n = static_cast<int>(ceil(log(sz) / log(2)));
47
48   // Block available
49   if (free_list[n].size() > 0)
50   {
51     pair<int, int> temp = free_list[n][0];
52
53     // Remove block from free list
54     free_list[n].erase(free_list[n].begin());
55     cout << "Memory from " << temp.first
56          << " to " << temp.second << " allocated" << "\n";
57     // map starting address with size to make deallocating easy
58     mp[temp.first] = temp.second - temp.first + 1;
59   }
60   else{
61     int i;
62     for (i = n + 1; i < length; i++)
63     {
64       // Find block size greater than request
65       if (free_list[i].size() != 0)  break;
66     }
67     // If no such block is found i.e., no memory block available
68     if (i == length){
69       cout << "Sorry, failed to allocate memory \n";
70     }
71
```

Listing 3: 分配

```
72      // If found
73      else
74      {
75        pair<int, int> temp;
76        temp = free_list[i][0];
77
78        // Remove first block to split it into halves
79        free_list[i].erase(free_list[i].begin());
80        i--;
81
82        for (; i >= n; i--){
83          // Divide block into two halves
84          pair<int, int> pair1, pair2;
85          pair1 = make_pair(temp.first,
86                  temp.first +(temp.second -temp.first) / 2);
87          pair2 = make_pair(temp.first +(temp.second - temp.first + 1) / 2,
88                  temp.second);
89          free_list[i].push_back(pair1);
90
91          // Push them in free list
92          free_list[i].push_back(pair2);
93          temp = free_list[i][0];
94
95          // Remove first free block to further split
96          free_list[i].erase(free_list[i].begin());
97        }
98        cout << "Memory from " << temp.first
99             << " to " << temp.second<< " allocated" << "\n";
100       mp[temp.first] = temp.second -
101       temp.first + 1;
102     }
103   }
104 }
```

Listing 4: 分配续