实验报告目录

01 实验概述

1.1 实验目的

通过编写模拟程序以加深对动态分区分配方式及其实现过程的理解。

1.2 实验要求

编写C程序，模拟实现动态分区存储管理方案。要求实现**首次/最佳**适应算法的内存块分配和回收，假设初始状态下，可用内存空间是一块连续区域，大小为102400B。要求实现的功能有：

1. 分配内存
2. 回收内存
3. 显示内存（显示空闲分区和已分配分区的情况）

02 实验设计与实现

2.1 实验设计概述

- 首次适应算法
  - 空闲分区链表按地址升序排列；
  - 分配：从头开始查找，找到第一个满足大小的空闲分区，分配给进程；
  - 回收：将回收的分区插入空闲分区链表中，若与前后空闲分区相邻，则合并；回收时需保持空闲分区链表仍按地址升序排列；
- 最佳适应算法
  - 空闲分区链表按分区大小升序排列；
  - 分配：从头开始查找，找到第一个满足大小的空闲分区，分配给进程；
  - 回收：将回收的分区插入空闲分区链表中，若与前后空闲分区相邻，则合并；回收时需保持空闲分区链表仍按分区大小升序排列；

2.2 数据结构与全局定义

由于动态内存分配需要对空闲内存块表、已分配内存块表频繁进行插入和删除操作，因而使用可$O(1)$增删的链表组织信息。

这里使用C++中的STL list实现本实验所需的链表功能，并定义如下固定长度内存分区结构体：

```
// 固定大小内存分区结构
struct Block
{
    string name;   // 进程名
    int startAddr; // 起始地址
    int size;      // 大小
    Block(string name, int startAddr, int size) :
        name(name), startAddr(startAddr), size(size) {}
};
```

定义本实验全局变量：

- 内存总空间大小 MAX_SIZE ；
- 空闲分区链表 freeList ；
- 已分配分区连边 allocList ；

```
const int MAX_SIZE = 102400;

list<Block> freeList;  // 空闲分区链表
list<Block> allocList; // 已分配分区链表
```

对链表进行初始化，将可用内存空间作为一整块空闲分区：

```
// 初始化，将可用内存空间作为一z整块空闲分区
void Init()
{
    freeList.clear();
    allocList.clear();
    freeList.push_back(Block("1", 0, MAX_SIZE));
}
```

2.3 首次适应算法实现

2.3.1 内存分配

首次适应算法的内存分配实现如下：

```
// 首次适应算法分配内存（空闲分区链表按照起始地址排序）
// 输入：申请空间名称和大小
// 输出：分配是否成功
bool FirstFitAlloc(string name, int size)
{
    // 遍历空闲分区链表，找到第一个满足要求的分区
    for (auto it = freeList.begin(); it != freeList.end(); it++)
    {
        if (it->size >= size)
        {
            // 创建分配分区
```

```
            Block allocBlock(name, it->startAddr, size);

            // 插入已分配分区链表，按照起始地址排序
            auto it2 = allocList.begin();
            while (it2 != allocList.end() && it2->startAddr < allocBlock.startAddr)
                it2++;
            allocList.insert(it2, allocBlock);

            // 更新空闲分区链表
            if (it->size == size)
            {
                // 分区大小相等，删除当前空闲分区
                freeList.erase(it);
            }
            else
            {
                // 分区大小不等，更新空闲分区，起始地址后移，大小减小
                it->startAddr += size;
                it->size -= size;
            }
            return true;
        }
    }

    return false;
}
```

2.3.2 内存回收

首次适应算法的内存回收实现如下：

```
// 首次适应算法回收内存
// 输入：回收空间名称
// 输出：与回收分区合并的空闲分区数，-1表示回收失败(未找到)
int FirstFitFree(string name)
{
    // 遍历已分配分区链表，找到要回收的分区
    for (auto it = allocList.begin(); it != allocList.end(); it++)
    {
        if (it->name == name)
        {
            // 创建空闲分区
            Block freeBlock(name, it->startAddr, it->size);
```

```
13
14              // 插入空闲分区链表，按照起始地址排序
15              auto it2 = freeList.begin();
16              while (it2 != freeList.end() && it2->startAddr < freeBlock.startAddr)
17                  it2++;
18
19              freeList.insert(it2, freeBlock);
20              it2--; // 将指针移动到新插入分区位置
21
22              // 更新已分配分区链表
23              allocList.erase(it);
24
25              // 合并空闲分区
26              // 仅可能与前后两个分区合并，
27              // 检查是否存在且地址是否连续
28              int mergeCount = 0;
29              auto itBefore = it2;
30              itBefore--; // 前一个分区
31              if (it2 != freeList.begin() &&
32                  itBefore->startAddr + itBefore->size == it2->startAddr)
33              {
34                  // 与前一个分区合并
35                  // 保留当前分区，删除前一个分区
36                  it2->size += itBefore->size;
37                  it2->startAddr = itBefore->startAddr;
38                  freeList.erase(itBefore);
39                  mergeCount++;
40              }
41
42              auto itAfter = it2;
43              itAfter++; // 后一个分区
44              if (itAfter != freeList.end() &&
45                  it2->startAddr + it2->size == itAfter->startAddr)
46              {
47                  // 与后一个分区合并
48                  // 保留当前分区，删除后一个分区
49                  it2->size += itAfter->size;
50                  freeList.erase(itAfter);
51                  mergeCount++;
52              }
53              return mergeCount;
54          }
55      }
56      return -1; // 未找到name对应的分区
57  }
```

2.4 最佳适应算法实现

2.4.1 内存分配

最佳适应算法的内存分配实现如下：

```
// 最佳适应算法分配内存（空闲分区链表按照大小排序）
// 输入：申请空间名称和大小
// 输出：分配是否成功
bool BestFitAlloc(string name, int size)
{
    // 遍历空闲分区链表，找到最小的满足要求的分区
    // 由于空闲分区链表按照大小排序，因此只需找到首个满足大小的分区即可
    auto it = freeList.begin();
    while (it != freeList.end() && it->size < size)
        it++;

    // 如果找到最佳分区
    if (it != freeList.end())
    {
        // 创建分配分区
        Block allocBlock(name, it->startAddr, size);

        // 插入已分配分区链表，按照起始地址排序
        // 即插入到第一个大小大于等于当前分区的分区之前
        auto it2 = allocList.begin();
        while (it2 != allocList.end() && it2->startAddr < allocBlock.startAddr)
            it2++;
        allocList.insert(it2, allocBlock); // 注意：insert插入it位置之前

        // 更新空闲分区链表
        if (it->size == size)
        {
            // 分区大小相等，删除当前空闲分区
            freeList.erase(it);
        }
        else
        {
            // 分区大小不等，更新空闲分区，起始地址后移，大小减小
            it->startAddr += size;
            it->size -= size;
```

```
        }
        return true;
    }

    return false;
}

```

2.4.2 内存回收

最佳适应算法的内存回收实现如下：

```
// 最佳适应算法回收内存
// 回收分区按照分区大小排序，合并后仍按照分区大小排序
// 输入：回收空间名称
// 输出：与回收分区合并的空闲分区数，-1表示回收失败(未找到)
int BestFitFree(string name)
{
    // 遍历已分配分区链表，找到要回收的分区
    for (auto it = allocList.begin(); it != allocList.end(); it++)
    {
        if (it->name == name)
        {

            // 创建空闲分区
            Block freeBlock(name, it->startAddr, it->size);

            // 删除已分配分区
            allocList.erase(it);

            int mergeCount = 0;
            // 首先检查是否可与前后两个分区合并
            for (auto it2 = freeList.begin(); it2 != freeList.end(); it2++)
            {
                // 如果it2是freeBlock的前一个分区
                if (it2->startAddr + it2->size == freeBlock.startAddr)
                {
                    // 将it2合并到freeBlock，删除it2
                    freeBlock.startAddr = it2->startAddr;
                    freeBlock.size += it2->size;
                    freeList.erase(it2);
                    mergeCount++;
                }

```

```
32
33                  // 如果it2是freeBlock的后一个分区
34                  if (it2->startAddr == freeBlock.startAddr + freeBlock.size)
35                  {
36                      // 将it2合并到freeBlock，删除it2
37                      freeBlock.size += it2->size;
38                      freeList.erase(it2);
39                      mergeCount++;
40                  }
41
42                  // 如果已经合并了两个分区，退出循环
43                  if (mergeCount == 2)
44                      break;
45              }
46
47              // 插入空闲分区链表，保持大小升序排序
48              auto it2 = freeList.begin();
49              while (it2 != freeList.end() && it2->size < freeBlock.size)
50                  it2++;
51              freeList.insert(it2, freeBlock);
52
53              return mergeCount;
54          }
55      }
56      return -1; // 未找到name对应的分区
57  }
```

2.5 内存分配与回收封装

实验需实现两种内存分配算法，为方便调用，将内存分配和回收封装为函数，使得根据输入选择，传入不同算法的函数指针即可实现不同算法的内存分配和回收。

2.5.1 内存分配封装

```
1  // 读入申请空间名称和大小，根据传入的分配函数指针分配内存
2  //  (分配函数指针可为FirstFitAlloc或BestFitAlloc)
3  void Alloc(bool (*AllocFunc)(string, int))
4  {
5      string name;
6      int size;
7      cout << "请输入申请空间名称 > ";
```

```
    cin >> name;
    cout << "请输入申请空间大小 > ";
    cin >> size;

    if (AllocFunc(name, size))
        cout << "分配成功!" << endl;
    else
        cout << "分配失败!" << endl;
}

```

2.5.2 内存回收封装

```
// 读入回收空间名称，根据传入的回收函数指针回收内存
//  (回收函数指针可为FirstFitFree或BestFitFree)
void Free(int (*FreeFunc)(string))
{
    string name;
    cout << "请输入回收空间名称 > ";
    cin >> name;

    int mergeCount = FreeFunc(name);
    if (mergeCount == -1)
    {
        cout << "回收失败!" << endl;
    }
    else
    {
        cout << "回收成功! 合并空闲分区数:  " << mergeCount << endl;
    }
}
```

2.6 打印内存状态

打印内存空闲分区和已分配分区的状态，实现如下：

```
void Show()
{
    // 打印空闲分区
    cout << "\n空闲分区: " << endl;
    cout << "编号  起始地址  大小(B)\n";

```

```
    cout.setf(ios::right);

    int idx = 0;
    for (auto it = freeList.begin(); it != freeList.end(); it++)
    {
        cout << setw(3) << ++idx
             << setw(9) << it->startAddr
             << setw(10) << it->size
             << endl;
    }
    cout << endl;

    // 打印已分配分区
    cout << "已分配分区: " << endl;
    cout << "名称  起始地址  大小(B)\n";
    for (auto it = allocList.begin(); it != allocList.end(); it++)
    {
        cout << setw(3) << it->name
             << setw(9) << it->startAddr
             << setw(10) << it->size
             << endl;
    }
}
```

2.7 主函数

初始化内存空间，根据用户选择设置Alloc和Free对应算法，循环读入用户输入，根据输入选择执行不同操作，实现如下：

```
int main()
{
    Init();

    string op; // string 避免异常输入
    do
    {
        cout << "==================== 动态分区模拟实验 ==================\n";
        cout << "选择分配算法(1.首次适应算法 2.最佳适应算法),\n";
        cout << "输入算法编号 > ";
        cin >> op;
    } while (op != "1" && op != "2");

```

```
    // 根据用户选择的分配算法，设置分配函数指针
    auto AllocFunc = (op == "1") ? FirstFitAlloc : BestFitAlloc;
    auto FreeFunc = (op == "1") ? FirstFitFree : BestFitFree;

    while (true)
    {
        cout << endl;
        cout << "=====================================================\n";
        cout << "选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),\n";
        cout << "请输入操作编号 > "; cin >> op;

        if (op == "1")
            Alloc(AllocFunc);
        else if (op == "2")
            Free(FreeFunc);
        else if (op == "3")
            Show();
        else if (op == "4")
            break;
        else
            cout << "输入错误，请重新输入!" << endl;
    }

    return 0;
}
```

03 实验测试与结果

3.1 首次适应算法测试

```
PS D:\files\Course\2023Fal-操作系统\lab\lab6>.\dynamic.exe
==================== 动态分区模拟实验 ==================
选择分配算法(1.首次适应算法 2.最佳适应算法),
输入算法编号 > 1

=======================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 3

空闲分区:
编号   起始地址   大小(B)
  1         0    102400

已分配分区:
名称   起始地址   大小(B)

=======================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 1
请输入申请空间名称 > s1
请输入申请空间大小 > 100
分配成功!

=======================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 3

空闲分区:
编号   起始地址   大小(B)
  1       100    102300

已分配分区:
名称   起始地址   大小(B)
 s1         0       100
```

```
35
36  ======================================================
37  选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
38  请输入操作编号 > 1
39  请输入申请空间名称 > s2
40  请输入申请空间大小 > 200
41  分配成功!
42
43  ======================================================
44  选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
45  请输入操作编号 > 1
46  请输入申请空间名称 > s3
47  请输入申请空间大小 > 300
48  分配成功!
49
50  ======================================================
51  选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
52  请输入操作编号 > 1
53  请输入申请空间名称 > s4
54  请输入申请空间大小 > 400
55  分配成功!
56
57  ======================================================
58  选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
59  请输入操作编号 > 3
60
61  空闲分区:
62  编号   起始地址   大小(B)
63    1      1000     101400
64
65  已分配分区:
66  名称   起始地址   大小(B)
67   s1         0        100
68   s2       100        200
69   s3       300        300
70   s4       600        400
71
72  ======================================================
73  选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
74  请输入操作编号 > 2
75  请输入回收空间名称 > s1
76  回收成功! 合并空闲分区数:  0
77
78  ======================================================
79  选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
```

```
请输入操作编号 > 3

空闲分区:
编号   起始地址   大小(B)
  1         0        100
  2      1000     101400

已分配分区:
名称   起始地址   大小(B)
 s2       100        200
 s3       300        300
 s4       600        400

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 2
请输入回收空间名称 > s3
回收成功! 合并空闲分区数:  0

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 3

空闲分区:
编号   起始地址   大小(B)
  1         0        100
  2       300        300
  3      1000     101400

已分配分区:
名称   起始地址   大小(B)
 s2       100        200
 s4       600        400

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 2
请输入回收空间名称 > s2
回收成功! 合并空闲分区数:  2

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 3

空闲分区:
```

```
编号   起始地址   大小(B)
  1         0       600
  2      1000    101400

已分配分区:
名称   起始地址   大小(B)
 s4       600       400

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 1
请输入申请空间名称 > s5
请输入申请空间大小 > 500
分配成功!

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 3

空闲分区:
编号   起始地址   大小(B)
  1       500       100
  2      1000    101400

已分配分区:
名称   起始地址   大小(B)
 s5         0       500
 s4       600       400

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 1
请输入申请空间名称 > s6
请输入申请空间大小 > 6000000
分配失败!

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 2
请输入回收空间名称 > s4
回收成功! 合并空闲分区数:  2

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 3
```

```
空闲分区:
编号   起始地址   大小(B)
  1       500     101900

已分配分区:
名称   起始地址   大小(B)
 s5         0        500

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 2
请输入回收空间名称 > s5
回收成功! 合并空闲分区数:  1

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 3

空闲分区:
编号   起始地址   大小(B)
  1         0     102400

已分配分区:
名称   起始地址   大小(B)

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 4
PS D:\files\Course\2023Fal-操作系统\lab\lab6>
```

3.2 最佳适应算法测试

```
PS D:\files\Course\2023Fal-操作系统\lab\lab6>.\dynamic.exe
==================== 动态分区模拟实验 ==================
选择分配算法(1.首次适应算法 2.最佳适应算法),
输入算法编号 > 2

```

```
==========================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 3

空闲分区:
编号   起始地址   大小(B)
  1         0    102400

已分配分区:
名称   起始地址   大小(B)

==========================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 1
请输入申请空间名称 > s1
请输入申请空间大小 > 100
分配成功!

==========================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 3

空闲分区:
编号   起始地址   大小(B)
  1       100    102300

已分配分区:
名称   起始地址   大小(B)
 s1         0       100

==========================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 1
请输入申请空间名称 > s2
请输入申请空间大小 > 200
分配成功!

==========================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 1
请输入申请空间名称 > s3
请输入申请空间大小 > 300
分配成功!

==========================================================
```

```
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 1
请输入申请空间名称 > s4
请输入申请空间大小 > 400
分配成功!

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 3

空闲分区:
编号   起始地址   大小(B)
  1      1000     101400

已分配分区:
名称   起始地址   大小(B)
 s1         0        100
 s2       100        200
 s3       300        300
 s4       600        400

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 2
请输入回收空间名称 > s1
回收成功! 合并空闲分区数:  0

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 3

空闲分区:
编号   起始地址   大小(B)
  1         0        100
  2      1000     101400

已分配分区:
名称   起始地址   大小(B)
 s2       100        200
 s3       300        300
 s4       600        400

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出),
请输入操作编号 > 2
```

```
请输入回收空间名称 > s3
回收成功！合并空闲分区数： 0

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出)，
请输入操作编号 > 3

空闲分区：
编号   起始地址   大小(B)
  1         0        100
  2       300        300
  3      1000     101400

已分配分区：
名称   起始地址   大小(B)
 s2       100        200
 s4       600        400

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出)，
请输入操作编号 > 1
请输入申请空间名称 > s5
请输入申请空间大小 > 50
分配成功！

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出)，
请输入操作编号 > 3

空闲分区：
编号   起始地址   大小(B)
  1        50         50
  2       300        300
  3      1000     101400

已分配分区：
名称   起始地址   大小(B)
 s5         0         50
 s2       100        200
 s4       600        400

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出)，
请输入操作编号 > 2
请输入回收空间名称 > s2
```

```
回收成功！合并空闲分区数： 2

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出)，
请输入操作编号 > 3

空闲分区：
编号   起始地址   大小(B)
  1        50        550
  2      1000     101400

已分配分区：
名称   起始地址   大小(B)
 s5         0         50
 s4       600        400

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出)，
请输入操作编号 > 1
请输入申请空间名称 > s6
请输入申请空间大小 > 6000000
分配失败！

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出)，
请输入操作编号 > 2
请输入回收空间名称 > s4
回收成功！合并空闲分区数： 2

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出)，
请输入操作编号 > 3

空闲分区：
编号   起始地址   大小(B)
  1        50     102350

已分配分区：
名称   起始地址   大小(B)
 s5         0         50

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出)，
请输入操作编号 > 2
请输入回收空间名称 > s5
```

```
回收成功！合并空闲分区数： 1

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出)，
请输入操作编号 > 3

空闲分区：
编号   起始地址   大小(B)
  1        0    102400

已分配分区：
名称   起始地址   大小(B)

=====================================================
选择操作(1.分配内存 2.回收内存 3.显示内存 4.退出)，
请输入操作编号 > 4
```

04 实验总结

本次实验编写动态分区模拟程序，分别实现了适应算法和最佳适应算法的内存分配和回收功能，对两种算法的特点和区别有了更深入的理解。在本实验的实现过程中，使用了C++中的STL list实现链表，使得链表的插入和删除操作可在$O(1)$完成，提高了程序的效率。