实验六：动态分区分配算法

姓名：汪江豪　学号：22121630　实验日期：2024.12.8

---

1. 实验环境：

实验设备：Lenovo Legion R7000P2021H
开发环境：VScode

2. 实验目的：

（1）深入理解操作系统中动态内存分配算法的工作原理和性能差异
（2）比较不同算法在内存利用率、分配成功率和内存碎片率等方面的表现
（3）观察和分析内存碎片的形成及其对系统性能的影响，理解如何选择合适的算法以最小化碎片。

3. 实验内容：

（1）分别实现首次适应算法、循环首次适应算法、最佳适应算法和最坏适应算法。
（2）假设初始状态下，可用的内存空间为 640 KB，编写程序在这 640KB 的空间上模拟上述动态分区分配算法的过程，其中，空闲分区通过空闲分区表来管理。作业请求序列如下：

- 作业 1 申请 130 KB 。
- 作业 2 申请 60 KB 。
- 作业 3 申请 100 KB 。
- 作业 2 释放 60 KB 。
- 作业 4 申请 200 KB 。
- 作业 3 释放 100 KB 。
- 作业 1 释放 130 KB 。
- 作业 5 申请 140 KB 。
- 作业 6 申请 60 KB 。
- 作业 7 申请 50 KB 。
- 作业 6 释放 60 KB 。

要求每次分配和回收后显示出空闲内存分区链的情况。
思考：这些分配算法主要适用于何种情况？

4. 实现代码：

4.1 分区结构体设计

```
// 分区结构体
struct Partition
{
    int size;         // 剩余分区大小
    int startAddress; // 起始地址
    bool free = true; // 是否空闲
    vector<PII> list;
```

```
};
```

4.2 分配算法设计：

```
// 首次适应算法
int firstFit(vector<Partition> &memory, string &name, int processSize)
{
    for (auto &part : memory)
        if (part.size >= processSize)
        {
            if (part.free)
                part.free = false;
            part.list.push_back({name, processSize});
            int tmp = part.size - processSize;
            if (tmp < 0)
                break;
            part.size = tmp;
            return part.startAddress;
        }
    return -1; // 没有足够的空间
}
// 循环首次适应算法
int nextFit(vector<Partition> &memory, string &name, int processSize, int
&nextIndex)
{
    for (int i = 0; i < memory.size(); i++)
    {
        int index = (i + nextIndex) % memory.size();
        auto &val = memory[index];
        if (val.size >= processSize)
        {
            if (val.free)
                val.free = false;
            val.list.push_back({name, processSize});
            val.free = false;
            int tmp = val.size - processSize;
            if (tmp < 0)
                break;
            val.size = tmp;
            nextIndex = index + 1;
            return val.startAddress;
        }
    }
    return -1;
}
```

```
// 最佳适应算法
int bestFit(vector<Partition> &memory, string &name, int processSize)
{
    int bestIndex = -1;
    int minSize = INT_MAX;
    for (int i = 0; i < memory.size(); i++)
    {
        if (memory[i].size >= processSize && memory[i].size < minSize)
        {
            bestIndex = i;
            minSize = memory[i].size;
        }
    }
    if (bestIndex != -1)
    {
        auto &val = memory[bestIndex];
        val.list.push_back({name, processSize});
        if (val.free)
            val.free = false;
        int tmp = val.size - processSize;
        if (tmp < 0)
            return -1;
        val.size = tmp;
        return val.startAddress;
    }
    return -1;
}

// 最坏适应算法
int worstFit(vector<Partition> &memory, string &name, int processSize)
{
    int worstIndex = -1;
    int maxSize = 0;
    for (int i = 0; i < memory.size(); i++)
    {
        if (memory[i].size >= processSize && memory[i].size > maxSize)
        {
            worstIndex = i;
            maxSize = memory[i].size;
        }
    }
    if (worstIndex != -1)
    {
```

```
        auto &val = memory[worstIndex];
        val.list.push_back({name, processSize});
        if (val.free)
            val.free = false;
        int tmp = val.size - processSize;
        if (tmp < 0)
            return -1;
        val.size = tmp;
        return val.startAddress;
    }
    return -1;
}
```

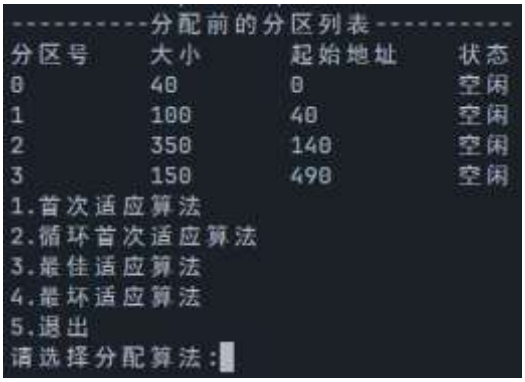
4.3 回收内存分配算法

```
//  回收内存分配算法
bool recycle(vector<Partition> &memory, string &name)
{
    for (auto &part : memory)
    {
        if (part.free)
            continue;
        for (auto &cur : part.list)
        {
            if (cur.x == name)
            {
                part.size += cur.y;
                part.list.erase(find(part.list.begin(), part.list.end(),
cur));
                if (part.list.empty())
                    part.free = true;
                return true;
            }
        }
    }
    return false;
}
```

5. 结果：

5.1 初始化状态

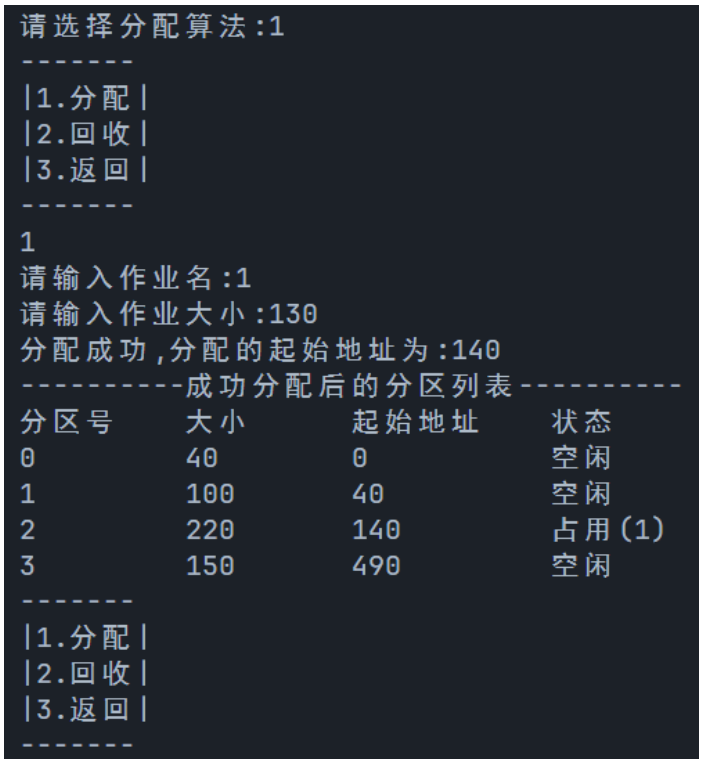
选择 1-首次适应算法，2-循环首次适应算法，3-最佳适应算法，4-最坏适应算法，5-直接退出



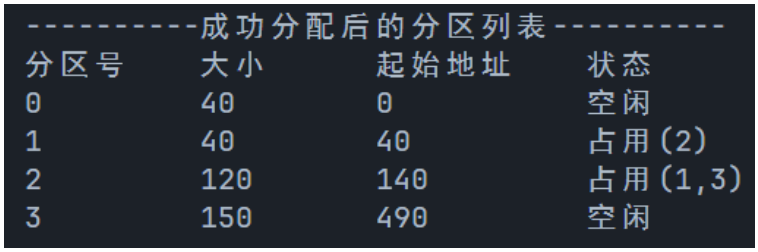
5.2 分配示例

选择分配算法后，可选择 1-分配，2-回收内存，3-返回。一个分配序列中可以选择多种不同分配算法。

选择分配后，输入作业名和作业大小后，即可按照指定算法分配内存，随后打印空闲分区列表的剩余空间。



按首次适应算法，成功分配作业 1-130KB,作业 2-60KB，作业 3-100KB 后



5.3 回收内存测试：

输入 2 选择回收内存，输入要回收的作业名即可完成回收。

```
-------
|1.分配|
|2.回收|
|3.返回|
-------
2
请输入要回收的作业名:2
回收成功!
----------回收后的分区列表---------
分区号      大小        起始地址      状态
0           40          0           空闲
1           100         40          空闲
2           120         140         占用(1,3)
3           150         490         空闲
```

可看到，作业 2 被占用的空间已经成功回收。

5.4 分配算法测试：

5.4.1 首次适应算法：

先后分配作业 a-300KB,作业 b-40KB,可以看到先后分配了分区 2，0 的空间，结果正确。

```
-------
|1.分配|
|2.回收|
|3.返回|
-------
1
请输入作业名:a
请输入作业大小:300
分配成功,分配的起始地址为:140
----------成功分配后的分区列表----------
分区号      大小        起始地址      状态
0           40          0           空闲
1           100         40          空闲
2           50          140         占用(a)
3           150         490         空闲
```

```
-------
|1.分配|
|2.回收|
|3.返回|
-------
1
请输入作业名:b
请输入作业大小:40
分配成功,分配的起始地址为:0
----------成功分配后的分区列表----------
分区号      大小        起始地址      状态
0           0           0           占用(b)
1           100         40          空闲
2           50          140         占用(a)
3           150         490         空闲
```

5.4.2 最坏适应算法：

尝试分配作业 a，大小 20KB，可以看到，选择了最大分区 2，结果正确。

```
请选择分配算法:4
-------
|1.分配|
|2.回收|
|3.返回|
-------
1
请输入作业名:a
请输入作业大小:20
分配成功,分配的起始地址为:140
----------成功分配后的分区列表----------
分区号      大小        起始地址      状态
0           40          0             空闲
1           100         40            空闲
2           330         140           占用(a)
3           150         490           空闲
-------
```

5.4.3 最佳适应算法：

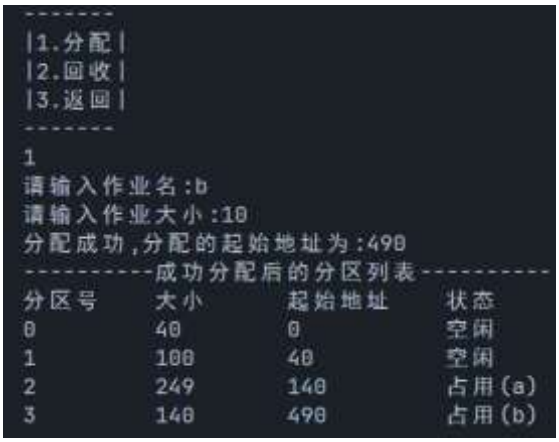
尝试分配作业 w，大小 21KB，可以看到，选择了最小的符合要求的分区 0，结果正确

```
请选择分配算法:3
-------
|1.分配|
|2.回收|
|3.返回|
-------
1
请输入作业名:w
请输入作业大小:21
分配成功,分配的起始地址为:0
----------成功分配后的分区列表----------
分区号      大小        起始地址      状态
0           19          0             占用(w)
1           100         40            空闲
2           350         140           空闲
3           150         490           空闲
-------
```

5.4.4 循环首次适应算法：

先分配作业 a-101KB,可以看到分区 2 被占用，再分配作业 b-10KB,可以看到分区 3 被占用，可以看到分配的分区指针从上次分配的地方开始检索，结果正确。

```
-------
|1.分配|
|2.回收|
|3.返回|
-------
1
请输入作业名:a
请输入作业大小:101
分配成功,分配的起始地址为:140
----------成功分配后的分区列表----------
分区号      大小        起始地址      状态
0           40          0             空闲
1           100         48            空闲
2           249         140           占用(a)
3           150         490           空闲
```



体会：

在本次动态分区分配算法的实验中，我使用 C++实现了首次适应算法、循环首次适应算法、最佳适应算法和最坏适应算法这四种分配算法。加深了我对于操作系统的内存管理机制的理解。让我深刻地意识到了 OS 内存分配机制的强大性。我对这四种分配算法的又去额点也有了更深刻的理解。

首次适应算法，优点是简单容易实现，复杂度低。缺点是容易在低地址部分产生许多小的空闲块，造成内存碎片。

循环首次适应算法，优点是可以更好地分散空闲块，减少内存碎片，不需要从头开始查找，提高查找效率。缺点是容易出现低地址和高地址部分使用不均衡。

最佳适应算法，优点是尽量减少了内存碎片，找到最小的满足要求的空闲块，提高了内存利用率。缺点是需要遍历整个空闲分区表，复杂度较高，容易产生大量小的空闲块，不利于大作业的分配，频繁的分割和合并可能导致效率低下。

最坏适应算法，优点是可以有效减少小碎片，因为总是选择最大的空闲块分配，对大作业分配较为有利。缺点是可能浪费较多内存，因为即使作业很小，也会分配最大的空闲块，随着时间推移，可能难以找到足够大的空闲块来满足大作业的需求。

总之，各个内存分配算法各有优劣，实际操作过程中需要灵活选择。