操作系统实验报告

班级：计科 4 班 姓名：陈昊天 学号：2021329600006

实验三：主存空间的分配与回收

一、 实验题目

采用可变式分区管理，使用最佳适应算法实现主存的分配与回收

二、 实验内容

主存是中央处理机能直接存取指令和数据的存储器。能否合理而有效地使用主存，在很大程度上将影响到整个计算机系统的性能。本实验采用可变式分区管理，使用首次或最佳适应算法实现主存空间的分配与回收。要求采用分区说明表进行。

三、 实验目的

通过本次实验，帮助学生理解在可变式分区管理方式下，如何实现主存空间的分配与回收。

提示：

(1) 可变式分区管理是指在处理作业过程中建立分区，使分区大小正好适合作业的需要，并且分区个数是可以调整的。当要装入一个作业时，根据作业需要的主存量，查看是否有足够的空闲空间，若有，则按需求量分割一部分给作业；若无，则作业等待。随着作业的装入、完成，主存空间被分割成许多大大小小的分区。有的分区被作业占用，有的分区空闲。例如，某时刻主存空间占用情况如图 1 所示。

0	操作系统（10KB）
10K	作业 1（10KB）
20K	作业 4（25KB）
45K	空闲区 1（20KB）
65K	作业 2（45KB）
110K	空闲区 2（146KB）
256K	

表 1 空闲区说明表

起始地址	长度	状态
45K	20KB	未分配
110K	146KB	未分配
		空表目
		空表目
		空表目
…	…	…

图 1　主存空间占用情况

为了说明哪些分区是空闲的，可以用来装入新的作业，必须要有一张**空闲区说明表**，如表 1 所示。

其中，起始地址指出各空闲区的主存起始地址，长度指出空闲区大小。

状态栏未分配指该栏目是记录的有效空闲区，空表目指没有登记信息。

由于分区个数不定，所以空闲区说明表中应有足够的空表目项，否则造成溢出，无法登记。

同样，再设一个**已分配区表**，记录作业或进城的主存占用情况。

(2)　当有一个新作业要求装入主存时，必须查空闲区说明表，从中找出一个足够大的空闲区。有时找到的空闲区可能大于作业需求量，这时应该将空闲区一分为二。一个分给作业，另一个仍作为空闲区留在空闲区表中。为了尽量减少由于分割造成的碎片，尽可能分配低地址部分的空闲区，将较大空闲区留在高地址端，以利于大作业的装入。为此在空闲区表中，按空闲区首地址从低到高进行登记。为了便于快速查找，要不断地对表格进行紧缩，即让“空表目”项留在表的后部。其分配框图如图 2 所示。

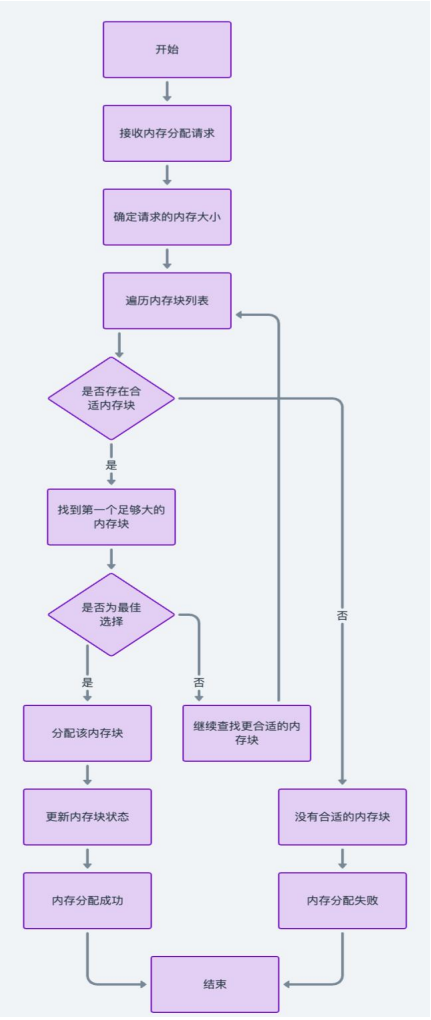

图 2　最佳适应算法内存分配框图

（3）当一个作业执行完时，作业所占用的分区应归还给系统。在归还时要考虑相邻空闲区合并的问题。作业的释放区与空闲区的邻接分一下 4 种情况考虑：

A. 释放区下邻（低地址邻接）空闲区；

B. 释放区上邻（高地址邻接）空闲区；

C. 释放区上下都与空闲区邻接；

D. 释放区与空闲区不邻接。

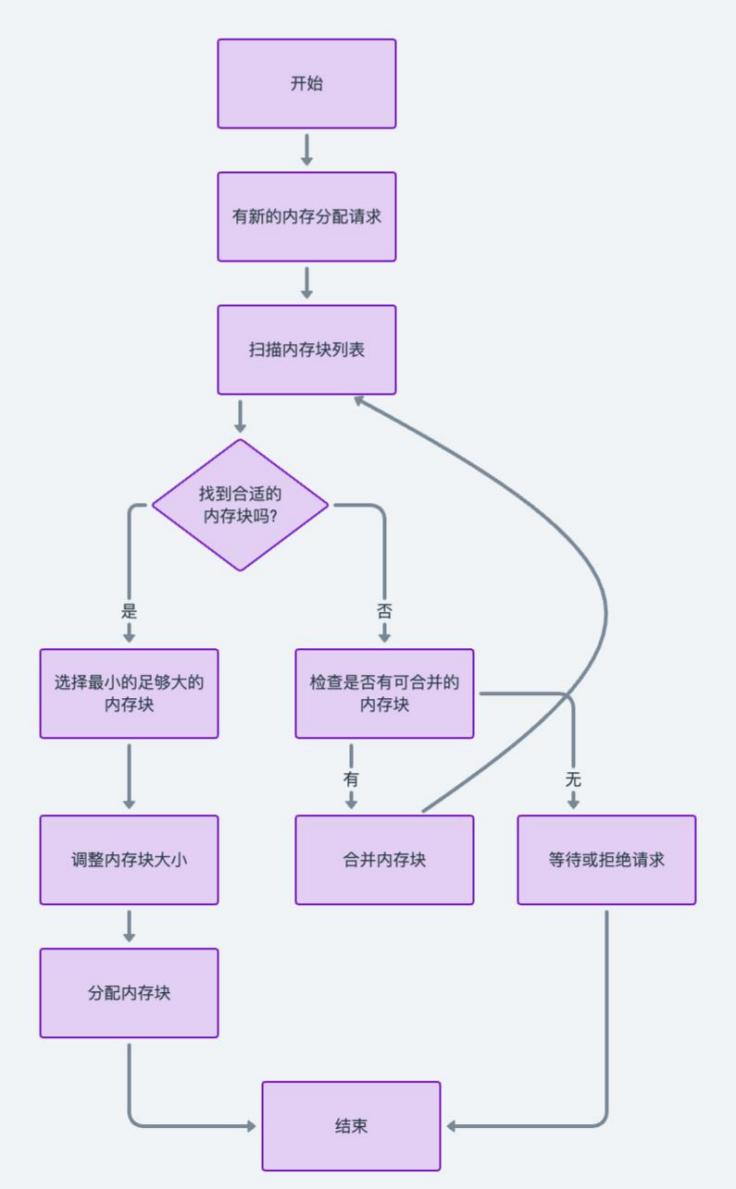

图 3 最佳适应算法回收框图

若采用最佳适应算法，则空闲区说明表中的空闲区按其大小排序。**有关最佳适应算法的分配和回收框图由学生自己给出。**

（4）请按最佳适应算法设计主存分配和回收程序。以图 1 作为主存当前使用的基础，初始化空闲区和已分配区说明表的值。设计一个作业申请队列以及作业完成后的释放顺序，实现主存的分配与回收。把空闲区说明表的变化情况以及各作业的申请、释放情况显示或打印出来。

为了说明哪些分区是空闲的，必须要有一张空闲区说明表，格式如下表所示：

起始地址	长度	状态
20K	20K	1
80K	50K	1
150K	100K	1
300K	30K	0（空表目）
600K	100K	1
…	…	空表目
…	…	…

四、**代码及运行结果分析**

4.1 代码

```
#include <algorithm>
#include <iostream>
#include <list>

struct MemoryBlock {
    int startAddress;
    int size;
    std::string status;  // "操作系统", "作业 X", "空闲区"

    MemoryBlock(int start, int sz, std::string st)
        : startAddress(start), size(sz), status(st) {}
};

bool compareMemoryBlock(const MemoryBlock& a, const MemoryBlock& b)
{
    return a.size < b.size;
}

void initializeMemory(std::list<MemoryBlock>& freeList,
                      std::list<MemoryBlock>& allocatedList) {
    // 内存容量为 256K

    // 已分配区

    allocatedList.push_back(MemoryBlock(0, 10, "操作系统"));

    allocatedList.push_back(MemoryBlock(10, 10, "作业 1"));
```

```
    allocatedList.push_back(MemoryBlock(20, 25, "作业 4"));

    allocatedList.push_back(MemoryBlock(65, 45, "作业 2"));


    // 空闲区

    freeList.push_back(MemoryBlock(45, 20, "空闲区"));

    freeList.push_back(MemoryBlock(110, 146, "空闲区"));
}

bool allocateMemory(std::list<MemoryBlock>& freeList,
                    std::list<MemoryBlock>& allocatedList, int size,
                    std::string jobName) {

    std::cout << "为" << jobName << "分配" << size << "空间: ";

    auto bestFit = freeList.end();


    // 找最佳适应块

    for (auto it = freeList.begin(); it != freeList.end(); ++it) {
        if (it->size >= size &&
            (bestFit == freeList.end() || it->size < bestFit->size))
{
            bestFit = it;
        }
    }

    if (bestFit != freeList.end()) {

        // 分配内存

        allocatedList.push_back(
            MemoryBlock(bestFit->startAddress, size, jobName));


        // 更新空闲区块

        bestFit->startAddress += size;
        bestFit->size -= size;

        if (bestFit->size == 0) {
            freeList.erase(bestFit);
        }
```

```
            std::cout << "分配成功" << std::endl;
            return true;
        }
        std::cout << "分配失败" << std::endl;
        return false;
    }

    void deallocateMemory(std::list<MemoryBlock>& freeList,
                          std::list<MemoryBlock>& allocatedList,
                          std::string jobName) {
        std::cout << "为" << jobName << "回收空间" << std::endl;
        for (auto it = allocatedList.begin(); it != allocatedList.end();
    ++it) {
            if (it->status == jobName) {
                int start = it->startAddress;
                int size = it->size;

                // 从已分配列表中删除
                allocatedList.erase(it);

                // 合并空闲区块
                for (auto fit = freeList.begin(); fit != freeList.end();)
    {
                    if (fit->startAddress + fit->size ==
                        start) {  // 空闲区块在回收区块前
                        size += fit->size;
                        start = fit->startAddress;
                        fit = freeList.erase(fit);
                    } else if (start + size ==
                               fit->startAddress) {  // 空闲区块在回收区块后
                        size += fit->size;
                        fit = freeList.erase(fit);
                    } else {
                        ++fit;
                    }
                }
```

```
            // 加入新空闲区块

            freeList.push_back(MemoryBlock(start, size, "空闲区"));

            freeList.sort([](const MemoryBlock& a, const MemoryBlock&
b) {
                return a.startAddress < b.startAddress;
            });
            break;
        }
    }
}

void printMemoryState(const std::list<MemoryBlock>& freeList,
                      const std::list<MemoryBlock>& allocatedList) {
    std::list<MemoryBlock> mergedList = freeList;
    mergedList.insert(mergedList.end(), allocatedList.begin(),
                      allocatedList.end());

    mergedList.sort([](const MemoryBlock& a, const MemoryBlock& b) {
        return a.startAddress < b.startAddress;
    });


    std::cout << "主存状态：" << std::endl;

    for (const auto& block : mergedList) {

        std::cout << "起始地址：" << block.startAddress << ", 长度："

                  << block.size << ", 状态：" << block.status <<
std::endl;
    }
    std::cout << std::endl;
}

int main() {
    std::list<MemoryBlock> freeList;
    std::list<MemoryBlock> allocatedList;


    // 初始化

    initializeMemory(freeList, allocatedList);
    printMemoryState(freeList, allocatedList);
```

```
    std::string command, jobName;
    int size;

    while (true) {
        std::cout
            << "输入命令（alloc <作业名> <大小> | dealloc <作业名> | exit）:
";
        std::cin >> command;

        if (command == "alloc") {
            std::cin >> jobName >> size;
            if   (!allocateMemory(freeList,   allocatedList,   size,
jobName)) {
                std::cout << "内存分配失败。" << std::endl;
            }
            printMemoryState(freeList, allocatedList);
        } else if (command == "dealloc") {
            std::cin >> jobName;
            deallocateMemory(freeList, allocatedList, jobName);
            printMemoryState(freeList, allocatedList);
        } else if (command == "exit") {
            break;
        } else {
            std::cout << "无效命令。" << std::endl;
        }
    }

    return 0;
}
```

4.2 运行结果分析

实验采用了可变式分区管理的方式，结合最佳适应算法进行主存空间的分配与回收。
设计的操作序列：
alloc 作业 3 20
dealloc 作业 2
dealloc 作业 1
dealloc 作业 4

dealloc 作业 3

设计的操作序列体现了 4 种情况考虑:

A. 释放区下邻（低地址邻接）空闲区； （作业 4）

B. 释放区上邻（高地址邻接）空闲区； （作业 2）

C. 释放区上下都与空闲区邻接； （作业 3）

D. 释放区与空闲区不邻接。（作业 1）

```
(base) nanmener@Haotians-MacBook-Pro 实验3 % ./"main"
主存状态:
起始地址: 0, 长度: 10, 状态: 操作系统
起始地址: 10, 长度: 10, 状态: 作业1
起始地址: 20, 长度: 25, 状态: 作业4
起始地址: 45, 长度: 20, 状态: 空闲区
起始地址: 65, 长度: 45, 状态: 作业2
起始地址: 110, 长度: 146, 状态: 空闲区

输入命令（alloc <作业名> <大小> | dealloc <作业名> | exit）: alloc 作业3 20
为作业3分配20空间: 分配成功
主存状态:
起始地址: 0, 长度: 10, 状态: 操作系统
起始地址: 10, 长度: 10, 状态: 作业1
起始地址: 20, 长度: 25, 状态: 作业4
起始地址: 45, 长度: 20, 状态: 作业3
起始地址: 65, 长度: 45, 状态: 作业2
起始地址: 110, 长度: 146, 状态: 空闲区

输入命令（alloc <作业名> <大小> | dealloc <作业名> | exit）: dealloc 作业2
为作业2回收空间
主存状态:
起始地址: 0, 长度: 10, 状态: 操作系统
起始地址: 10, 长度: 10, 状态: 作业1
起始地址: 20, 长度: 25, 状态: 作业4
起始地址: 45, 长度: 20, 状态: 作业3
起始地址: 65, 长度: 191, 状态: 空闲区

输入命令（alloc <作业名> <大小> | dealloc <作业名> | exit）: dealloc 作业1
为作业1回收空间
主存状态:
起始地址: 0, 长度: 10, 状态: 操作系统
起始地址: 10, 长度: 10, 状态: 空闲区
起始地址: 20, 长度: 25, 状态: 作业4
起始地址: 45, 长度: 20, 状态: 作业3
起始地址: 65, 长度: 191, 状态: 空闲区

输入命令（alloc <作业名> <大小> | dealloc <作业名> | exit）: dealloc 作业4
为作业4回收空间
主存状态:
起始地址: 0, 长度: 10, 状态: 操作系统
起始地址: 10, 长度: 35, 状态: 空闲区
起始地址: 45, 长度: 20, 状态: 作业3
起始地址: 65, 长度: 191, 状态: 空闲区

输入命令（alloc <作业名> <大小> | dealloc <作业名> | exit）: dealloc 作业3
为作业3回收空间
主存状态:
起始地址: 0, 长度: 10, 状态: 操作系统
起始地址: 10, 长度: 246, 状态: 空闲区
```

五、心得体会

通过本次实验，我对操作系统中的主存管理有了更深刻的理解。实验中采用的可变式分区管理和最佳适应算法，是理解动态存储分配策略的重要环节。我认识到，在实际应用中，有效地管理主存空间对于提高系统性能至关重要。通过对分区大小和数量的动态调整，能够更灵活地应对不同作业的内存需求，从而优化了内存的使用效率。

通过编写和调试分配与回收程序，我深入体会到了编程实践在理论知识学习中的价值。实验过程中不仅需要理解理论，还要将其应用于实际编程中，这对我的编程能力和逻辑思维能力是一次重大的挑战与提升。实验中出现的各种情况，如内存分配失败、空闲区与作业的邻接处理等，都需要细致的逻辑判断和算法实现，这些经验对我未来的学习和工作都有重要意义。

实验让我更加明白了计算机系统中资源管理的复杂性和重要性。主存作为有限的资源，如何高效利用并减少碎片化问题，是操作系统设计中的核心问题之一。通过这次实验，我对操作系统的设计和实现有了更全面的了解，也激发了我对深入学习操作系统原理的兴趣。