北京交通大学编译原理实验报告

姓名：程维森

学号：21231264

一. 实验内容介绍

这个实验涉及到虚拟内存管理的一些概念，主要包括页表、TLB（翻译后备缓冲器）以及页面置换等。虚拟内存是计算机中的一种技术，它允许程序在似乎有无限大的内存空间中运行，而实际上，只有部分数据和指令被加载到物理内存中。

具体来说，这个实验包含以下关键概念和步骤：

1. 虚拟地址到物理地址的映射：

- 通过读取文件中的逻辑地址，将其分解为页号和偏移量。
- 使用页号查找页表，获取物理帧号。
- 将物理帧号和偏移量组合成物理地址。

2. TLB 的使用：
- TLB 是一种高速缓存，存储了虚拟地址到物理地址的映射。在查找页表之前，先查找 TLB。
- 如果 TLB 命中，可以直接从 TLB 中获取物理帧号，否则需要继续查找页表。

3. 页面置换：
- 如果页表中未找到所需的页号，就需要从磁盘上读取数据（页面置换）。
- 在本例中，readFromDisk 函数负责从磁盘文件中读取数据，并将其加载到物理内存中。

4. 性能统计：
- 统计页面错误（Page Faults）和 TLB 命中（TLB Hits）的次数，以便评估虚拟内存管理的性能。

这个实验通过模拟虚拟内存管理的各个步骤，帮助理解计算机系统中关于虚拟内存的基本原理和概念。

二. 实验内容分析

1. 文件读取和内存映射：
- readFromDisk 函数用于从磁盘文件 BACKING_STORE.bin 中读取数据。这是模拟实际计算机系统中从磁盘加载数据到物理内存的操作。
- 通过 fseek 设置文件指针，以读取相应页号的数据，并将其加载到物理内存（PM 数组）中。

2. 查找页面：
- findPage 函数负责将逻辑地址转换为物理地址。
- 首先，将逻辑地址分解为页号和偏移量。
- 然后，在 TLB 中查找，如果找到，直接得到物理帧号。否则，继续在页表（PT 数组）中查找。
- 如果在页表中找到，得到物理帧号。否则，调用 readFromDisk 从磁盘加载数据。

3. TLB 的使用和页面置换：
- TLB 是一个缓存，存储了最近访问的页号到物理帧号的映射，以提高地址翻译的速度。
- 如果 TLB 命中，可以直接获得物理帧号。否则，需要从页表中查找。
- 如果页表中未找到，说明发生了页面错误，需要进行页面置换，调用 readFromDisk 从磁盘加载数据，并更新页表。

4. 性能统计和结果输出：
- 统计页面错误和 TLB 命中的次数，以及计算缺页率和 TLB 命中率。
- 将每个逻辑地址的虚拟地址、物理地址和数值输出到文件 output.txt。

5. 主函数：
- 打开输入文件，循环读取逻辑地址，并调用 findPage 函数进行地址转换。
- 最后，输出缺页率和 TLB 命中率。

总体而言，这个实验通过模拟虚拟内存管理的流程，包括 TLB 的使用和页面置换，帮助理解计算机系统中关于虚拟内存的基本原理和概念。

三. 实验逻辑分析

三. 实验逻辑分析：

1. readFromDisk 函数解析：

int readFromDisk(int pageNum, char *PM, int *OF)

- 通过 fseek 设置文件指针，将文件指针移动到对应页的位置，以读取页面数据。
- 使用 fread 从磁盘文件读取一页的数据到缓冲区 buffer。
- 将读取的数据写入物理内存数组 PM 中，根据 OF 指示的偏移量。
- 更新 OF，表示下一个可用的物理内存页。

2. findPage 函数解析：

int findPage(int logicalAddr, char *PT, TLB *tlb, char *PM, int *OF, int *pageFaults, int *TLBhits, std::ofstream &outputFile)

- 解析逻辑地址，得到页号 pageNum 和偏移量 offset。
- 在 TLB 中查找是否有对应页号的物理帧号，如果有，则 TLB 命中，直接使用 TLB 中的物理帧号。
- 如果 TLB 未命中，在页表 PT 中查找是否有对应页号的物理帧号。
- 如果页表中找到，得到物理帧号，更新 TLB。
- 如果页表未找到，发生缺页，调用 readFromDisk 函数加载数据，并更新页表和 TLB。
- 计算物理地址 index，读取物理内存中的数据，并输出到文件 outputFile。

3. 主函数解析：

int main(int argc, char *argv[])

- 打开输入文件和输出文件，定义相关变量和数据结构。
- 通过循环读取输入文件中的逻辑地址，调用 findPage 函数进行地址转换和数据加载。
- 统计页面错误次数和 TLB 命中次数。
- 计算缺页率和 TLB 命中率。
- 将结果输出到文件 output.txt。

总体而言，该实验的逻辑流程清晰，首先通过 readFromDisk 函数从磁盘加载数据到物理内存，然后通过 findPage 函数进行逻辑地址到物理地址的转换，同时处理 TLB 的使用和页面置换。最终，主函数统计性能指标并将结果输出。

四. 实验代码详解

4.1readFromDisk 函数

readFromDisk 函数的目的是从磁盘文件中读取数据，然后将数据加载到物理内存中。以下是该函数的逐行讲解：

```
int readFromDisk(int pageNum, char *PM, int *OF)
{
    char buffer[BUFFER_SIZE];
    memset(buffer, 0, sizeof(buffer));
    std::ifstream BS("BACKING_STORE.bin", std::ios::binary);
    if (!BS.is_open())
    {
        std::cerr << "文件打开失败" << std::endl;
        exit(0);
    }
    if (BS.seekg(pageNum * PHYS_MEM_SIZE).fail())
    {
        std::cerr << "fseek错误" << std::endl;
    }

    if (!BS.read(buffer, PHYS_MEM_SIZE))
    {
        std::cerr << "fread错误" << std::endl;
    }

    for (int i = 0; i < PHYS_MEM_SIZE; i++)
    {
        PM[(*OF) * PHYS_MEM_SIZE + i] = buffer[i];
    }

    (*OF)++;

    return (*OF) - 1;
}
```

逐行解释：

1. char buffer[BUFFER_SIZE];：创建一个缓冲区 buffer，用于存储从磁盘文件中读取的数据，大小为 BUFFER_SIZE。

2. memset(buffer, 0, sizeof(buffer));：使用 memset 函数将缓冲区清零，确保没有旧数据残留。

3. std::ifstream BS("BACKING_STORE.bin", std::ios::binary);：以二

进制模式打开名为 "BACKING_STORE.bin" 的文件，并创建输入文件流 BS。

4. if (!BS.is_open()) {...}: 检查文件是否成功打开，如果打开失败，输出错误信息并退出程序。

5. if (BS.seekg(pageNum * PHYS_MEM_SIZE).fail()) {...}: 将文件指针移到对应页号的位置。pageNum * PHYS_MEM_SIZE 计算了文件中页的起始位置。

6. if (!BS.read(buffer, PHYS_MEM_SIZE)) {...}: 从文件中读取 PHYS_MEM_SIZE 个字节的数据到缓冲区 buffer 中。

7. for (int i = 0; i < PHYS_MEM_SIZE; i++) {...}: 将缓冲区中的数据逐个加载到物理内存中，位置为 (*OF) * PHYS_MEM_SIZE + i，其中 (*OF) 是页表的打开帧数。

8. (*OF)++;: 增加页表的打开帧数，表示已经加载了一页的数据。

9. return (*OF) - 1;: 返回加载到物理内存的帧号，即打开帧数减一。

4.2findPage 函数

```
int findPage(int logicalAddr, char *PT, TLB *tlb, char *PM, int *OF, int *pageFaults, int *TLBhits, std::ofstream &outputFile)
{
    unsigned char mask = 0xFF;
    unsigned char offset;
    unsigned char pageNum;
    bool TLBhit = false;
    int frame = 0;
    int value;
    int newFrame = 0;

    outputFile << "虚拟地址: " << logicalAddr << "\t";

    pageNum = (logicalAddr >> 8) & mask;
    offset = logicalAddr & mask;

    for (int i = 0; i < TLB_SIZE; i++)
    {
        if (tlb->TLBpage[i] == pageNum)
        {
            frame = tlb->TLBframe[i];
            TLBhit = true;
            (*TLBhits)++;
        }
    }

    if (!TLBhit)
    {
        if (PT[pageNum] == -1)
        {
            newFrame = readFromDisk(pageNum, PM, OF);
            PT[pageNum] = newFrame;
            (*pageFaults)++;
        }
        frame = PT[pageNum];
        tlb->TLBpage[tlb->ind] = pageNum;
        tlb->TLBframe[tlb->ind] = frame;
        tlb->ind = (tlb->ind + 1) % TLB_SIZE;
    }
    int index = ((unsigned char)frame * PHYS_MEM_SIZE) + offset;
    value = *(PM + index);
    outputFile << "物理地址: " << index << "\t数值: " << static_cast<int>(value) << std::endl;
    return 0;
}
```

逐行解释:

1. unsigned char mask = 0xFF;: 创建一个无符号字符型的掩码，所有位均为 1。

2. unsigned char offset;, unsigned char pageNum;: 定义无符号字符型的变量 offset 和 pageNum，用于存储逻辑地址的偏移量和页号。

3. bool TLBhit = false;: 定义布尔型变量 TLBhit，表示 TLB 是否命中。

4. int frame = 0;, int value;, int newFrame = 0;: 定义整型变量

frame（表示物理帧号）、value（表示从物理内存读取的值）和 newFrame（表示新加载的物理帧号）。

5. outputFile << "虚拟地址: " << logicalAddr << "\t";: 将虚拟地址写入输出文件。

6. pageNum = (logicalAddr >> 8) & mask;, offset = logicalAddr & mask;: 解析逻辑地址，获取页号和偏移量。

7. for (int i = 0; i < TLB_SIZE; i++): 在 TLB 中遍历，查找是否有对应页号的物理帧号。

8. if (tlb->TLBpage[i] == pageNum): 如果找到匹配的页号。

9. frame = tlb->TLBframe[i];, TLBhit = true;, (*TLBhits)++;: 更新物理帧号，标记 TLB 命中，并增加 TLB 命中次数。

10. if (!TLBhit): 如果 TLB 未命中。

11. if (PT[pageNum] == -1): 如果页表中未找到对应页号的物理帧号。

12. newFrame = readFromDisk(pageNum, PM, OF);: 调用 readFromDisk 函数加载数据，获取新的物理帧号。

13. PT[pageNum] = newFrame;, (*pageFaults)++;: 更新页表，增加缺页次数。

14. frame = PT[pageNum];: 获取页表中的物理帧号。

15. tlb->TLBpage[tlb->ind] = pageNum;, tlb->TLBframe[tlb->ind] = frame;, tlb->ind = (tlb->ind + 1) % TLB_SIZE;: 更新 TLB

16. int index = ((unsigned char)frame * PHYS_MEM_SIZE) + offset;: 计算物理地址。

17. value = *(PM + index);: 从物理内存中读取数据。

18. outputFile << "物理地址：" << index << "\t 数值：" << static_cast<int>(value) << std::endl;: 将物理地址和数值写入输出文件。

4.3 比较函数

为了更好的比对自己的结果与正确答案，给出两个 python 函数来验证答案的完备性：

文件 1 (`extract_numbers.py`):

```python
import re

def extract_numbers(input_filename, output_filename):
    with open(input_filename, 'r') as input_file:
        lines = input_file.readlines()

    with open(output_filename, 'w') as output_file:
        for line in lines:
            # 使用正则表达式提取数字（包括负数）
            numbers = [int(match.group()) for match in re.finditer(r'-?\d+', line)]

            # 保证每行有三个数字
            if len(numbers) == 3:
                output_line = ','.join(map(str, numbers)) + '\n'
                output_file.write(output_line)
            else:
                print(f"Warning: Skipped line with less than three numbers: {line.strip()}")

# 用法示例
input_filename = 'lab05/output.txt'  # 你的输入文件名
output_filename = 'lab05/out.txt'  # 你的输出文件名

extract_numbers(input_filename, output_filename)
```

这个文件定义了一个函数 `extract_numbers`，它从一个文本文件中读取内容，提取每行中的数字（可能包括负数），确保每行有三个数字，并将提取的数字写入一个新的文本文件。这个函数使用了正则表达式来匹配数字，并在输出文件中每行使用逗号分隔。如果某一行的数字少于三个，则输出一个警告信息。

使用示例：

```python
input_filename = 'lab05/output.txt'  # 输入文件名
output_filename = 'lab05/out.txt'  # 输出文件名
extract_numbers(input_filename, output_filename)
```

文件 2 (`compare_files.py`):

```python
import re
def compare_files(file1, file2):
    flag = 1
    with open(file1, 'r') as f1, open(file2, 'r') as f2:
        lines1 = f1.readlines()
        lines2 = f2.readlines()

    # 确保两个文件行数相同
    if len(lines1) != len(lines2):
        print("行数不同，文件不匹配")
        return

    for i, (line1, line2) in enumerate(zip(lines1, lines2), 1):
        # 提取每行的数字（可能为负数）
        numbers1 = [int(match.group()) for match in re.finditer(r'-?\d+', line1)]
        numbers2 = [int(match.group()) for match in re.finditer(r'-?\d+', line2)]

        # 比较数字是否相同
        if numbers1 != numbers2:
            print(f"第 {i} 行不同:")
            print(f"文件1: {line1.strip()}")
            print(f"文件2: {line2.strip()}")
            print()
            flag = 0
    if flag == 1:
        print("完全一致！！！恭喜完成实验！！！")

# 用法示例
file1 = '/Users/sksx085/Desktop/各种实验报告等狗屎/OSHomework/exp5/lab05/out.txt'  # 你的文件1
file2 = '/Users/sksx085/Desktop/各种实验报告等狗屎/OSHomework/exp5/lab05/standard answer.txt'  # 你的文件2
```

这个文件定义了一个函数 `compare_files`，它比较两个文本文件的内容是否完全相同。如果两个文件的内容完全相同，输出一条消息，否则，它会逐行比较两个文件的内容，找出不同之处并输出。这个函数使用了一个简单的标志（`flag`）来表示是否找到不同之处。如果两个文件的行数不同，会输出一条相应的消息。

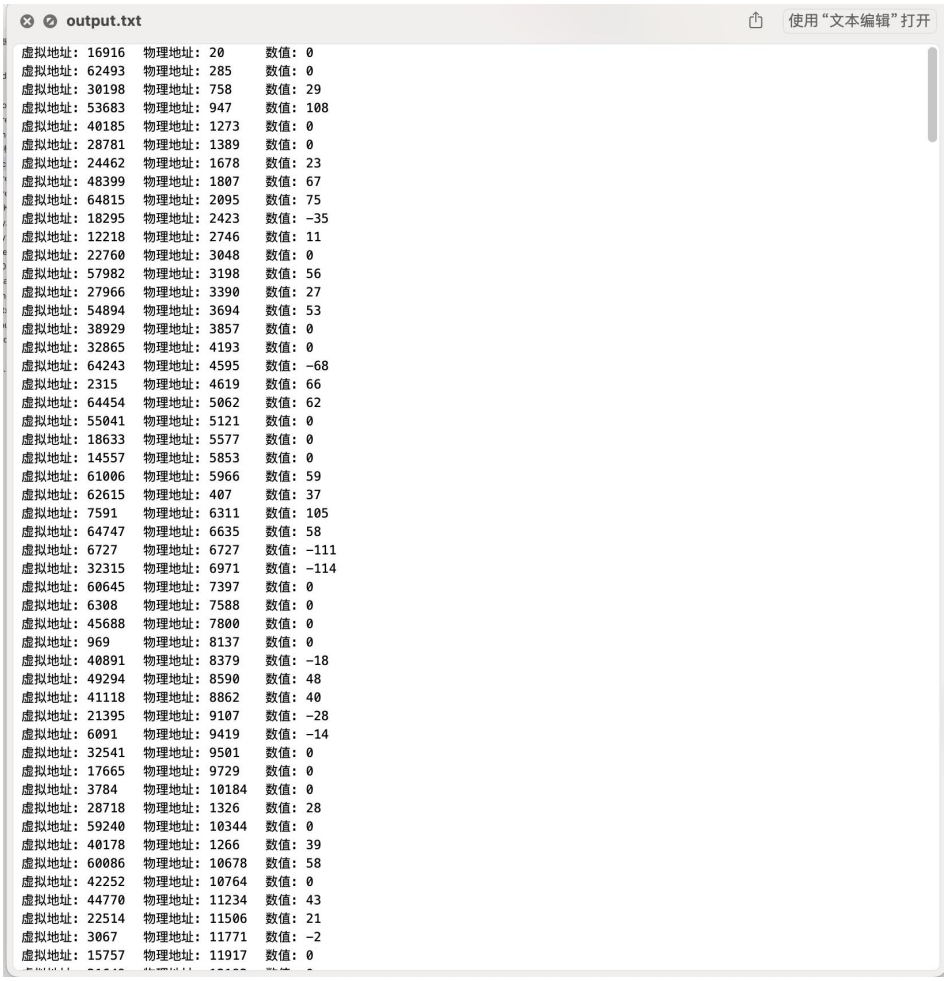
使用示例:

```python
file1 = 'lab05/out.txt'  # 第一个文件名
file2 = 'lab05/standard answer.txt'  # 第二个文件名
compare_files(file1, file2)
```

总体而言，这两个文件的目的是在提取数字后，通过比较文件的方式检查生成的输出文件与标准答案文件是否相同。

五. 实验结果展示

这是代码输出的结果:


output.txt
使用“文本编辑”打开
虚拟地址: 16916 物理地址: 20 数值: 0
虚拟地址: 62493 物理地址: 285 数值: 0
虚拟地址: 30198 物理地址: 758 数值: 29
虚拟地址: 53683 物理地址: 947 数值: 108
虚拟地址: 40185 物理地址: 1273 数值: 0
虚拟地址: 28781 物理地址: 1389 数值: 0
虚拟地址: 24462 物理地址: 1678 数值: 23
虚拟地址: 48399 物理地址: 1807 数值: 67
虚拟地址: 64815 物理地址: 2095 数值: 75
虚拟地址: 18295 物理地址: 2423 数值: -35
虚拟地址: 12218 物理地址: 2746 数值: 11
虚拟地址: 22760 物理地址: 3048 数值: 0
虚拟地址: 57982 物理地址: 3198 数值: 56
虚拟地址: 27966 物理地址: 3390 数值: 27
虚拟地址: 54894 物理地址: 3694 数值: 53
虚拟地址: 38929 物理地址: 3857 数值: 0
虚拟地址: 32865 物理地址: 4193 数值: 0
虚拟地址: 64243 物理地址: 4595 数值: -68
虚拟地址: 2315 物理地址: 4619 数值: 66
虚拟地址: 64454 物理地址: 5062 数值: 62
虚拟地址: 55041 物理地址: 5121 数值: 0
虚拟地址: 18633 物理地址: 5577 数值: 0
虚拟地址: 14557 物理地址: 5853 数值: 0
虚拟地址: 61006 物理地址: 5966 数值: 59
虚拟地址: 62615 物理地址: 407 数值: 37
虚拟地址: 7591 物理地址: 6311 数值: 105
虚拟地址: 64747 物理地址: 6635 数值: 58
虚拟地址: 6727 物理地址: 6727 数值: -111
虚拟地址: 32315 物理地址: 6971 数值: -114
虚拟地址: 60645 物理地址: 7397 数值: 0
虚拟地址: 6308 物理地址: 7588 数值: 0
虚拟地址: 45688 物理地址: 7800 数值: 0
虚拟地址: 969 物理地址: 8137 数值: 0
虚拟地址: 40891 物理地址: 8379 数值: -18
虚拟地址: 49294 物理地址: 8590 数值: 48
虚拟地址: 41118 物理地址: 8862 数值: 40
虚拟地址: 21395 物理地址: 9107 数值: -28
虚拟地址: 6091 物理地址: 9419 数值: -14
虚拟地址: 32541 物理地址: 9501 数值: 0
虚拟地址: 17665 物理地址: 9729 数值: 0
虚拟地址: 3784 物理地址: 10184 数值: 0
虚拟地址: 28718 物理地址: 1326 数值: 28
虚拟地址: 59240 物理地址: 10344 数值: 0
虚拟地址: 40178 物理地址: 1266 数值: 39
虚拟地址: 60086 物理地址: 10678 数值: 58
虚拟地址: 42252 物理地址: 10764 数值: 0
虚拟地址: 44770 物理地址: 11234 数值: 43
虚拟地址: 22514 物理地址: 11506 数值: 21
虚拟地址: 3067 物理地址: 11771 数值: -2
虚拟地址: 15757 物理地址: 11917 数值: 0

这是比对的结果

```
第 678 行不同：
文件1: 25450,60266,24
文件2: 25450,60522,24

第 683 行不同：
文件1: 33645,60525,0
文件2: 33645,60781,0

第 727 行不同：
文件1: 25498,60314,24
文件2: 25498,60570,24

第 735 行不同：
文件1: 30273,58689,0
文件2: 30273,58945,0

第 743 行不同：
文件1: 4542,58558,4
文件2: 4542,58814,4

第 757 行不同：
文件1: 34117,60741,0
文件2: 34117,60997,0

第 758 行不同：
文件1: 55555,59651,64
文件2: 55555,59907,64

第 783 行不同：
文件1: 57641,60969,0
文件2: 57641,61225,0

第 799 行不同：
文件1: 17375,61407,-9
文件2: 17375,61663,-9

第 834 行不同：
文件1: 18774,61526,18
文件2: 18774,61782,18

第 846 行不同：
文件1: 10054,59974,9
文件2: 10054,60230,9

第 864 行不同：
文件1: 31018,61738,30
文件2: 31018,61994,30

第 868 行不同：
文件1: 18904,61656,0
文件2: 18904,61912,0

第 906 行不同：
文件1: 62166,62166,60
文件2: 62166,62422,60

第 913 行不同：
文件1: 31114,61834,30
文件2: 31114,62090,30

第 929 行不同：
文件1: 49920,58880,0
文件2: 49920,59136,0

第 968 行不同：
文件1: 34070,60694,33
文件2: 34070,60950,33

第 976 行不同：
文件1: 57751,61079,101
文件2: 57751,61335,101

第 984 行不同：
文件1: 11983,59599,-77
文件2: 11983,59855,-77

(base) sksx085@sksx085deMacBook-Pro exp5 % /Users/sksx085/anaconda3/bin/python /Users/sksx085/Desktop/各种实验报告等狗屎/OSHomework/exp5/lab05/copy.py
Warning: Skipped line with less than three numbers: 缺页率 = 0.244
Warning: Skipped line with less than three numbers: TLB命中率 = 0.054
(base) sksx085@sksx085deMacBook-Pro exp5 % /Users/sksx085/anaconda3/bin/python /Users/sksx085/Desktop/各种实验报告等狗屎/OSHomework/exp5/lab05/copy2.py
(base) sksx085@sksx085deMacBook-Pro exp5 % /Users/sksx085/anaconda3/bin/python /Users/sksx085/Desktop/各种实验报告等狗屎/OSHomework/exp5/lab05/copy2.py
Traceback (most recent call last):
  File "/Users/sksx085/Desktop/各种实验报告等狗屎/OSHomework/exp5/lab05/copy2.py", line 49, in <module>
    compare_files(file1, file2)
  File "/Users/sksx085/Desktop/各种实验报告等狗屎/OSHomework/exp5/lab05/copy2.py", line 42, in compare_files
    if flag == 1:
       ^^^^
UnboundLocalError: cannot access local variable 'flag' where it is not associated with a value
(base) sksx085@sksx085deMacBook-Pro exp5 % /Users/sksx085/anaconda3/bin/python /Users/sksx085/Desktop/各种实验报告等狗屎/OSHomework/exp5/lab05/copy2.py
完全一致！！！恭喜完成实验！！！
(base) sksx085@sksx085deMacBook-Pro exp5 %
```

前面是错误信息：

错误原因：

```
int index = ((unsigned char)frame * PHYS_MEM_SIZE) + offset;

value = *(PM + index);

outputFile << "物理地址: " << index << "\t 数值: " << static_cast<int>(value) << std::endl;

return 0;
```

这个代码中，忘记将 index 强制转换为 int 而是采取了 index 小于 0 时+65536 的措施

六．不同内存管理方法

内存管理是操作系统中一个关键的组成部分，负责有效地分配和释放计算机内存。以下是一些不同的内存管理方法：

1. 单一连续区域内存管理：
   - 描述： 整个内存区域被看作一个单一的连续区域，分为内核空间和用户空间。
   - 优点： 实现简单，容易管理。
   - 缺点： 内存碎片可能导致浪费，不够灵活。
2. 分区内存管理：
   - 描述： 内存被划分为多个固定大小或可变大小的分区，每个分区用于存放一个进程。
   - 优点： 降低了内存碎片，能够同时运行多个进程。
   - 缺点： 外部碎片仍然可能存在。
3. 页式内存管理：
   - 描述： 内存被划分为固定大小的页面，进程被划分为相同大小的页面帧。进程的页面可以分散存放在内存中。
   - 优点： 减少了内存浪费，更灵活。
   - 缺点： 可能会有页面抖动。
4. 段式内存管理：
   - 描述： 内存被划分为不同长度的段，每个段用于存放一个逻辑单元。
   - 优点： 更好地反映了程序的逻辑结构
   - 缺点： 内部碎片可能存在。
5. 段页式内存管理：
   - 描述： 结合了段式和页式的优点，内存被划分为不同长度的段，每个段包含多个页面。
   - 优点： 兼顾了段式和页式的优点，提高了内存的利用率。
6. 动态内存管理：
   - 描述： 操作系统在运行时动态地分配和释放内存。
   - 优点： 更加灵活，能够适应不同程序的内存需求。
   - 缺点： 管理复杂，容易出现内存泄漏或溢出。
7. 伙伴系统：
   - 描述： 将内存划分为大小相等的块，每个块都是 2 的幂次方大小。通过合并和拆分来管理内存。
   - 优点： 减少了外部碎片。
   - 缺点： 可能存在一些内部碎片。

每种内存管理方法都有其适用的场景和优缺点，选择合适的方法取决于系统的需求和特性。不同的操作系统和应用程序可能会选择不同的内存管理策略。

七．代码与仓库

详细请参考：https://github.com/sksx085/OSHomework