# 《操作系统原理》

# 实习报告

| | |
|---|---|
| 班级 | 191211 |
| 学号 | 2021100156 |
| 姓名 | 孙鹤轩 |
| 院系 | 计算机学院 |
| 专业 | 计算机科学于技术 |
| 指导老师 | 张求明 |
| 完成日期 | 2023.05.03 |

# 实验一：多级队列调度算法

## 一、 程序功能及设计思路

**程序功能：**

实现多级队列调度算法，其中对 RQ1 采用轮转法，时间片 q=7；对 RQ2 采用短进程优先调度算法。在控制台输出调度过程和周转时间

**设计思路：**

首先定义进程块 PCB 结构体，创建进程队列 RQ1，RQ2，对进程队列进行初始化，设计两种算法函数，在 main 函数中开始进程调度。


## 二、 数据结构及算法设计

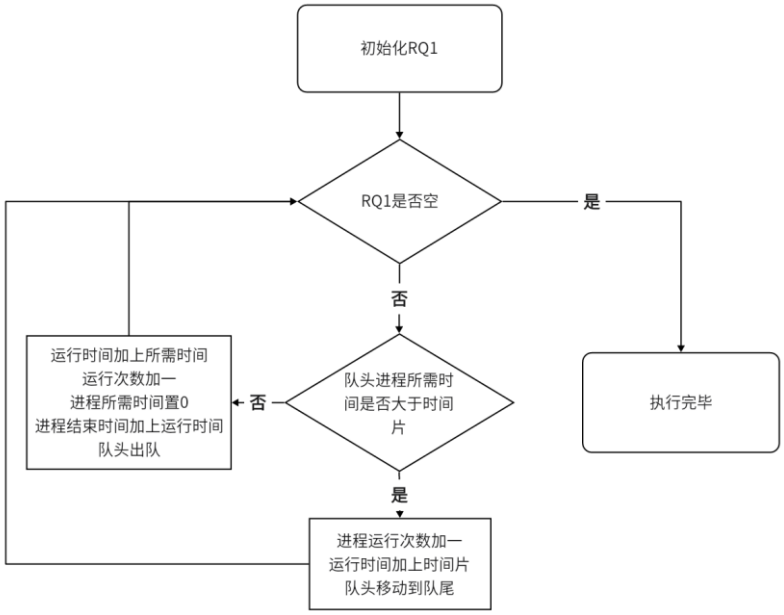
**PCB 结构体定义如下：**

```
struct PCB {
    std::string name = "";
    int need_time = 0;//进程运行所需时间
    int start_time = 0;//进程开始运行的时间
    int end_time = 0;//进程结束运行的时间
    int first_start_time = 0;//第一次开始运行的时间
    int run_time = 0;//已经运行的时间
    int wait_time = 0;//已等待时间
    int count = 0;//运行次数
};
```

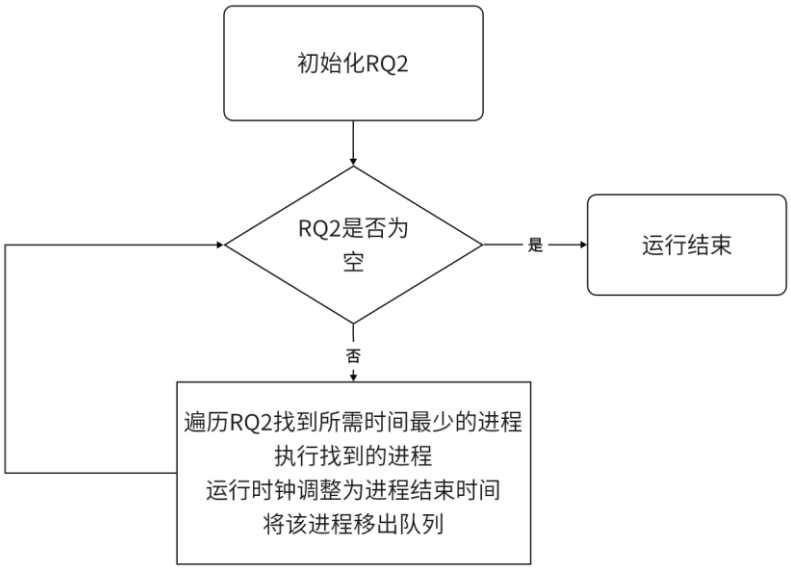
**采用链表组成进程队列，定义如下：**

```
std::list<PCB> RQ1, RQ2;
```

**算法设计如下：**

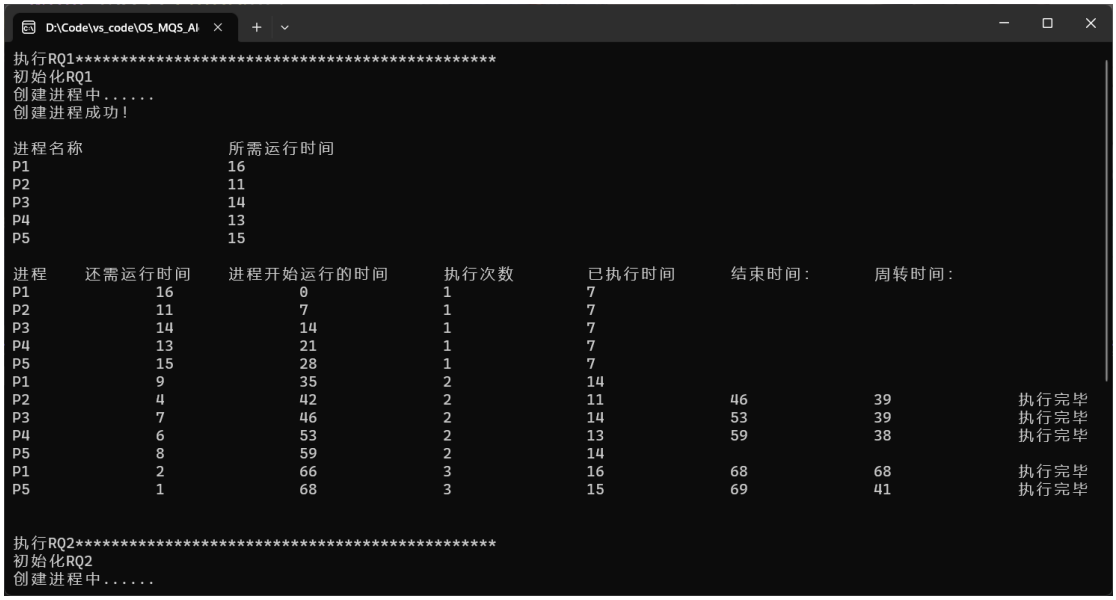


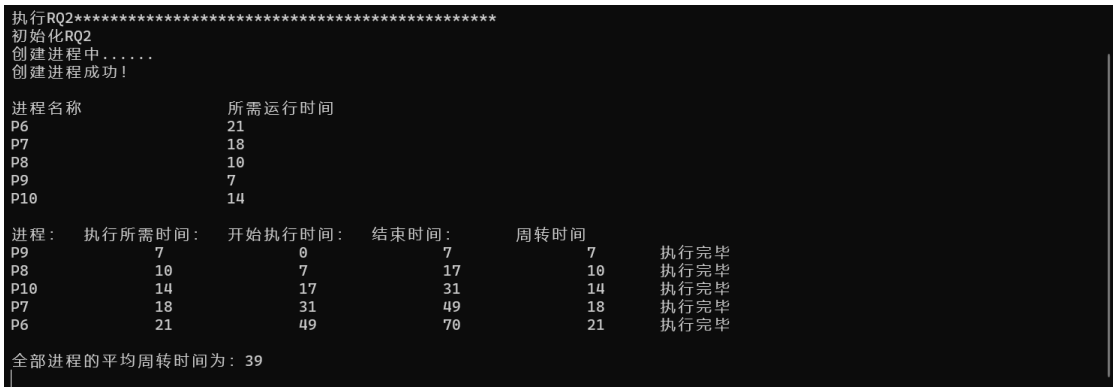

1 轮状算法流程图




2 短进程优先调度算法流程图

## 三、 程序运行情况




## 四、 实习心得

通过这次实习我对进程的调度有了更深的理解，并且自己实现了两种调度算法，用实践方法理解了轮转法和短进程优先两种调度算法。

在编程过程中我发现，对一个算法的执行要有良好的数据结构支撑，例如PCB 进程块结构体可以很好的维护进程，也方便管理进程，使用进程队列可以很好的对多个进程进行集中管理分配，这也是操作系统中很常见的管理办法，对此我对操作系统不再是雾里看花，有了更进一步的认识。

# 实验二：银行家算法

## 一、 程序功能及设计思路

**程序功能：**

通过 c++模拟实现银行家算法，在控制台输出调度结果。

**设计思路：**

首先宏定义进程个数和资源种类，定义一维数组 Available[m]，其中

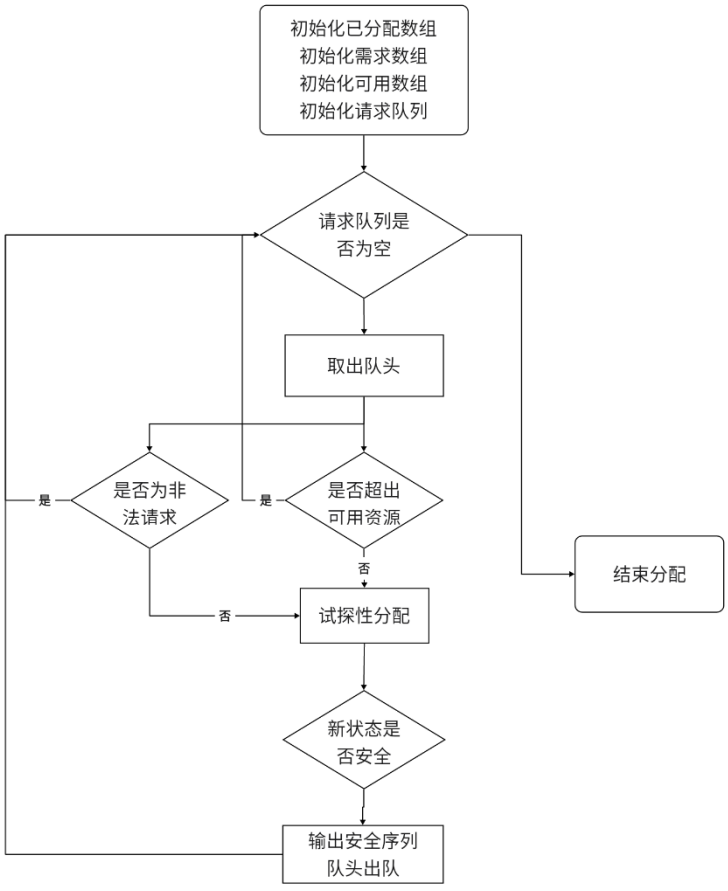
Available[i]表示第 i 种资源的可使用量，定义二维数组 Alloc[n][m],Need[n][m]，其中 Alloc[i][j]表示第 i 个进程第 j 类资源已分配的数量，Need[i][j]表示第 i 个进程第 j 类资源的需求数量。最后在 main 函数中执行银行家算法，输出结果。

## 二、 数据结构及算法设计

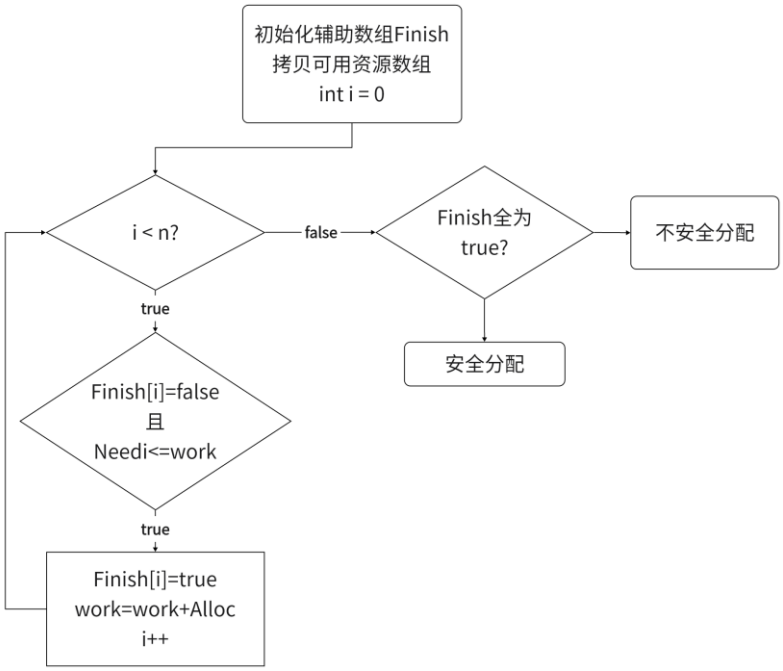
**数据结构定义如下：**

```
int Available[m + 1]{};//资源可使用数组
int Alloc[n + 1][m + 1]{ {} };//资源已分配数组
int Need[n + 1][m + 1]{ {} };//资源需求数组
std::vector<int> path;//可执行序列
struct Node
{
    int name;//进程ID
    int val[m + 1];//进程请求资源序列
};
```

**算法设计如下：**

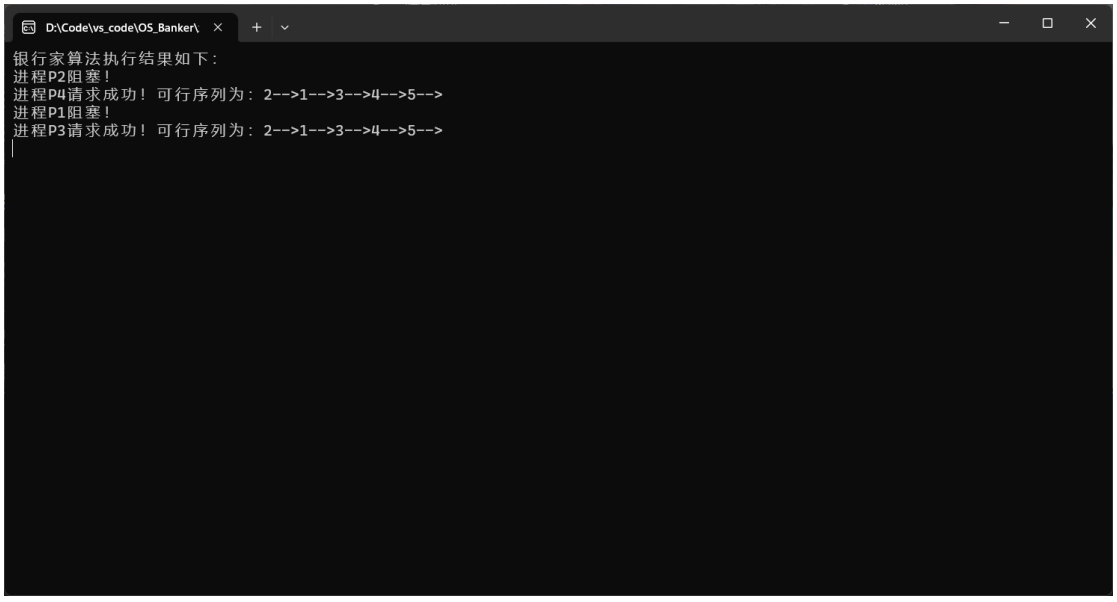



3 银行家算法流程图




4 安全状态判别算法

## 三、 程序运行情况




## 四、 实习心得

通过这次实习，让我对操作系统如何进行资源分配有了更深的理解，银行家算法是一个中规中矩的算法，它在很大程度上解决了死锁问题，当时频繁的进行安全状态检查会导致算法效率很低。

在编程过程中，算法的抽象以及数据结构的处理都对我的编程能力有了很大的提升，深深认识到算法在实际应用中的重要性，以及学习到了如何将算法应用到实际问题中。

# 实验三：动态分区式存贮区管理

## 一、 程序功能及设计思路

### 程序功能：

通过 c++模拟实现主存管理，提供首次，最佳，最坏三种放置策略选项，在控制台输出分配结果

**设计思路：**

定义枚举类型 Placement 用来枚举三种放置策略提供菜单功能；
定义 rd 分区描述器结构体；
宏定义申请的内存大小并计算节数；
定义空闲区队列以及使用区队列头节点
设计 Request 申请函数和 Realese 释放函数
在主程序中从文件中读取数据
执行选择的放置算法
输出结果

## 二、 数据结构及算法设计

**相关数据结构定义如下：**

```
enum Placement {//三种策略定义
    FIRST,//首次
    BEST,//最佳
    WOREST,//最坏
    EXIT//DEFALT
};
struct rd {//分区描述器
    int flag;  //分配标志，空闲为0
    int size;   //分区大小
    rd* next;   //指向下一个空闲分区，对已分配区：此项为零。
    rd() { flag = -1; size = -1; next = NULL; }//构造函数
    rd(int _flag , int _size , rd *_next) : flag(_flag) , size(_size) , next(_next){}//
带参构造
};
rd* Free , * Used;//分别用来存空闲队列和已占用队列
rd* block;//申请到的资源
```

**算法设计：**

**请求函数设计如下：{**

**选择放置算法->**

**首次放置算法：**空闲区队列按照物理地址的顺序，找到第一个符合需求的地址空间，分配给申请

**最佳放置算法：**空闲区队列按照容量大小递增排序，找到第一个符合需求的地址空间

**最坏放置算法：**空闲区队列按照容量大小递减排序，找到第一个符合需求的地址空间

根据放置策略更新空闲区队列和已使用队列

}

释放函数设计如下：{

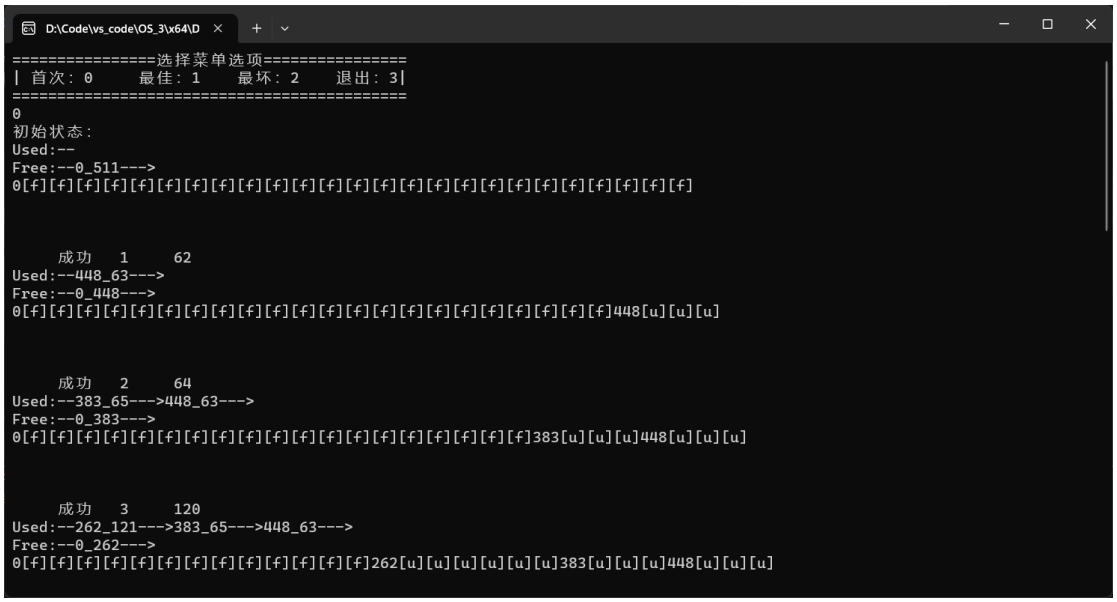
根据要释放的地址值在以使用队列中找到指定位置

在空闲区队列中检查有无高低相邻空闲区

更新空闲区队列和使用队列

}

## 三、 程序运行情况




## 四、 实习心得

通过这次实习我更深入的了解了操作系统中主存的分配过程和分配策略，学习并实现了三种放置策略以及空闲区回收策略，让我对操作系统的工作过程进一步有了深入了解。

在编程过程中，因为使用的是链表数据结构，编程策略与以往有不同，需要不断地操作指针来实现目的，这让我对底层数据结构有了更清晰的认知，对编程能力有了很大的提升。

# 实验四：请求分页系统中的置换算法

## 一、 程序功能及设计思路

**程序功能：**

根据指令生成算法生成 320 条指令；

执行三种页面置换算法并计算其命中率

在控制台输出指令序列和命中率计算结果

**设计思路：**

定义页面队列中的页面结构体 paper；

根据指令序列产生算法生成指令序列；

设计三种页面置换算法函数 FIFO,LRU,Optimal;

在主函数执行三种置换算法

在控制台输出指令序列和三种置换策略的命中率

## 二、 数据结构及算法设计

### 相关数据结构定义如下：

```
const int instruct_num = 320;//指令数量
int instruct_arr[instruct_num + 6];//指令集合
struct paper {//页面结构体，用来组成页面队列
    int instruct = 0;
    int time = 0;
    paper* next = NULL;
};
```

### 算法设计如下：

FIFO{

创建双向链表 FIFO 缓存，缓存块大小为 block_size。

初始化 Head 和 Tail 指向新 paper 节点。

对于每个指令 cur_instruct：

初始化命中标志 hit 为 false。

遍历双向链表 FIFO 缓存，如果找到 cur_instruct，则将命中标志 hit 设为 true。

如果没有命中：

增加缺页次数 failure_times。

如果 Head 所指的 paper 节点的指令数量大于等于 block_size，则将 Head 指向链表中下一个节点。

否则，增加 Head 所指节点的指令数量。

创建一个新的 paper 节点，将 Tail 指向新的节点，并将 cur_instruct 存储在新的节点中。

返回缺页率 1.0 减去失败次数 failure_times 除以指令总数 instruct_num 的比率。

}

LRU{

创建双向链表 LRU 缓存，缓存块大小为 block_size。

初始化 Head 和 Tail 指向新 paper 节点，初始化时钟 clock 为 999。

对于每个指令 cur_instruct：

时钟 clock 减 1。

初始化命中标志 hit 为 false。

遍历双向链表 LRU 缓存，如果找到 cur_instruct，则将命中标志 hit 设为 true，并刷新该节点的时钟为当前时钟 clock。

如果没有命中：

增加缺页次数 failure_times。

如果 Head 所指的 paper 节点的指令数量大于等于 block_size，则找到最久未使用的节点 t，将 t 的指令替换为 cur_instruct，并将 t 的时钟刷新为当前时钟 clock。

否则，增加 Head 所指节点的指令数量。

创建一个新的 paper 节点，将 Tail 指向新的节点，并将 cur_instruct 和当前时钟 clock 存储在新的节点中。

返回缺页率 1.0 减去失败次数 failure_times 除以指令总数 instruct_num 的比率。

}

Opetimal{

创建双向链表 Optimal 缓存，缓存块大小为 block_size。

初始化 Head 和 Tail 指向新 paper 节点。

对于每个指令 cur_instruct：

初始化命中标志 hit 为 false。

遍历双向链表 Optimal 缓存，如果找到 cur_instruct，则将命中标志 hit 设为 true。

如果没有命中：

增加缺页次数 failure_times。

如果 Head 所指的 paper 节点的指令数量大于等于 block_size，则对于每个缓存块 p，找到 p 请求还有多久会被用到，并将时间存储在 p 的 time 域中。

找到最久不被用到的缓存块 t，将 t 的指令替换为 cur_instruct。

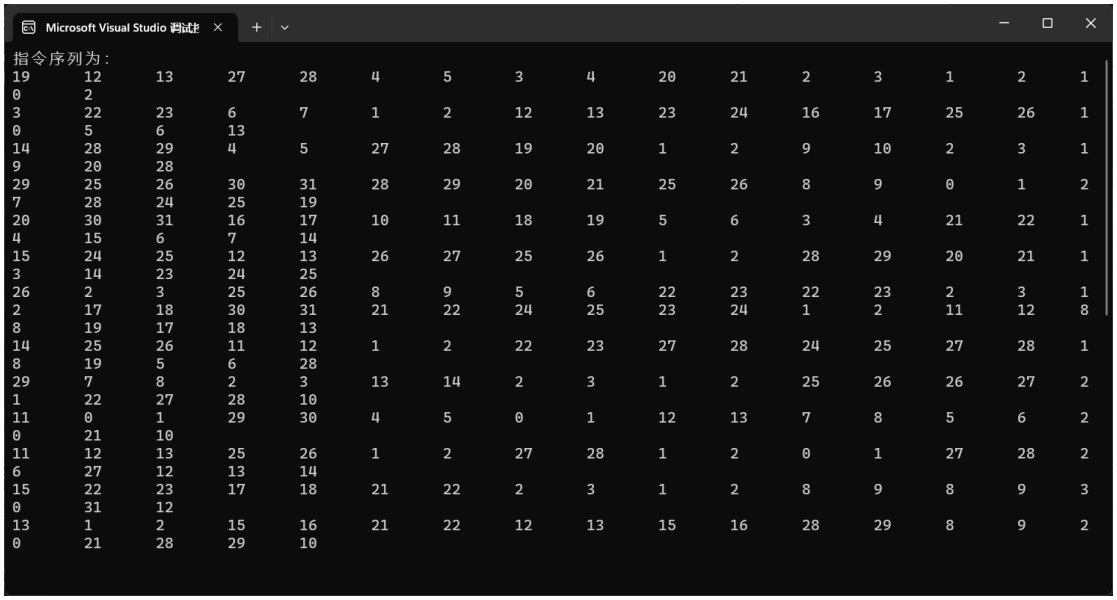
否则，增加 Head 所指节点的指令数量。

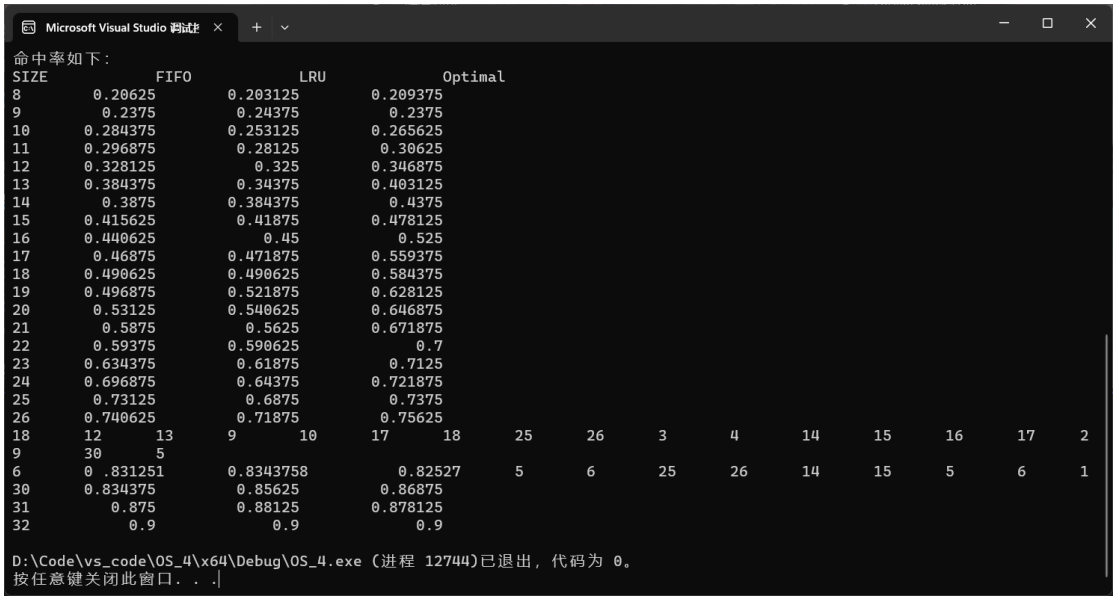
创建一个新的 paper 节点，将 Tail 指向新的节点，并将 cur_instruct 存储在新的节点中。

返回缺页率 1.0 减去失败次数 failure_times 除以指令总数 instruct_num 的比率。

}

# 三、 程序运行情况






## 四、 实习心得

通过这次实习我了解了操作系统中分页式系统中的页面置换策略，并且通过c++实现了三种置换策略，对操作系统的运行模式由浅入深的进行了学习。

在编程过程中依然是使用链表数据结构进行编程，数据结构有些类似双向链表，复习了双向链表的组成形态和实现过程，再一次感受到了双向链表的优越性。