一、实验源程序

头文件


```
#pragma once //编译器预编译命令，once：仅编译一次，常出现在头文件中。
             //因为同一头文件会在许多源文件中多次引用。如果没有指定编译一次，则编译时出现重定义错误
#ifndef _SCHEDULING_
#define _SCHEDULING_

#include <iostream>
#include <stdlib.h>
using namespace std;

enum Tag { UNSHD, SHD }; //枚举Tag标记是否被调度过，

/*
作业信息的数据结构PCB
*/
struct PCB
{
    int     Num;            //作业号
    size_t  arrtime;        //到达时间
    size_t  sertime;        //服务时间
    size_t  begtime;        //开始时间
    size_t  endtime;        //完成时间
    size_t  turntime;       //周转时间
    float   weighttime;     //带权周转时间
    PCB *   next;           //指向下个节点的指针
    Tag     tag;            //标记是否被调度过

    PCB(int n = 0, size_t a = 0, size_t s = 0)      //给这几个变量赋初值
        :Num(n), arrtime(a), sertime(s), begtime(0), endtime(0)
        , turntime(0), weighttime(0), next(NULL), tag(UNSHD)
    {}
};
```

```
/*
创建调度类
*/
class scheduling
{
public:
    scheduling() :_curtime(0), _tasknum(0)
    {
        _head = new PCB();
    }

    //先来先服务算法
    void FIFS()
    {
        if (empty())//判断是否有任务
        {
            cout << "没有任务";
            exit(-1);
        }
        _clear();  //清理缓存，可重复计算
        _sort_t(); //按到达时间排序
        PCB* cur = _head->next;
        while (NULL != cur)
        {
            if (_curtime < cur->arrtime)
            {
                _curtime = cur->arrtime;
            }
            cur->begtime = _curtime;
            cur->endtime = _curtime + cur->sertime;     //完成时间等于开始时间加服务时
间
            cur->turntime = cur->endtime - cur->arrtime;//周转时间=完成时间-到达时间
            cur->weighttime = (float)cur->turntime / (float)cur->sertime;  //带权周转
时间=周转时间/服务时间
            cur->tag = SHD; //标记为已经服务
            _curtime += cur->sertime;
            cur = cur->next;
        }
    }
```

```
//短作业优先

void Short()
{
    if (empty())
    {
        cout << "没有任务";
        exit(-1);
    }
    _clear();  //清理缓存，可重复计算
    _sort_t(); //按到达时间排序

    PCB* cur = _head->next;
    while (NULL != cur)
    {
        if (_curtime < cur->arrtime)
        {
            _curtime = cur->arrtime;
        }
        cur->begtime = _curtime;
        cur->endtime = _curtime + cur->sertime;       //完成时间等于开始时间加服务时间
        cur->turntime = cur->endtime - cur->arrtime;    //周转时间=完成时间-到达时间
        cur->weighttime = (float)cur->turntime / (float)cur->sertime; //带权周转时间=周转时间/服务时间
        cur->tag = SHD; //标记为已经服务
        _curtime += cur->sertime;
        cur = cur->next;

        //将该进程调度完的时刻已经到达的进程按短作业优先排序
        _sort_l(cur, _curtime);  //从该进程开始进行短作业排序

    }
}
```

```
//接收输入的作业
    void Init_task()
    {
        int tasknum = 0;
        size_t id = 0;
        size_t atime = 0;
        size_t stime = 0;
        cout << "请输入任务的个数:";
        cin >> tasknum;
        for (int i = 0; i<tasknum; i++)
        {
            cout << "请分别输入任务的编号,到达时间,运行时间:";
            cin >> id >> atime >> stime;
            push(id, atime, stime);
        }
    }

    void Push()
    {
        size_t id = 0;
        size_t atime = 0;
        size_t stime = 0;
        cout << "请分别输入任务的编号,到达时间,运行时间:";
        cin >> id >> atime >> stime;
        push(id, atime, stime);
    }
    /*
    输出函数
    */
    void Print()
    {
        if (empty())
            return;
        PCB* cur = _head->next;
        printf("-------------------------------------------------------------------
------- \n");
        printf("进程号 | 到达时间 | 服务时间 | 开始时间 | 完成时间 | 周转时间 | 带权周
转时间 \n");
        while (NULL != cur)
        {
            cout << "-------------------------------------------------------------
------------" << endl;
            printf("%4d    | %6d    |%8d   |%9d | %7d   |%8d   | %0.2f \n", cur->Num,
cur->arrtime, cur->sertime, cur->begtime, cur->endtime, cur->turntime,
```

```
cur->weighttime);
            cur = cur->next;
        }
        printf("-------------------------------------------------------------------------
------- \n");
    }


protected:
    bool empty()
    {
        return _tasknum == 0;
    }

    bool push(int n, size_t a, size_t s) //插入到链表尾部
    {
        PCB * newtask = new PCB(n, a, s);
        PCB * cur = _head;
        while (NULL != cur->next)
            cur = cur->next;
        cur->next = newtask;
        _tasknum++;
        return true;
    }

    void _clear()
    {
        if (empty())
            return;
        PCB* cur = _head->next;
        while (NULL != cur)
        {
            cur->begtime = 0;
            cur->endtime = 0;
            cur->turntime = 0;
            cur->weighttime = 0;
            cur->tag = UNSHD;
            cur = cur->next;
        }
        _curtime = 0;
    }
```

```
// 按照到达时间排序
    void _sort_t()
    {
        if (empty() || _tasknum == 1)
            return;
        PCB* prev = _head->next;
        PCB* cur = prev->next;
        for (int i = 0; i< _tasknum - 1; i++)
        {
            for (int j = 0; j<_tasknum - i - 1; j++)
            {
                if (prev->arrtime > cur->arrtime)
                {
                    _Swap(prev, cur);
                }
                prev = cur;
                cur = cur->next;
            }
            prev = _head->next;
            cur = prev->next;
        }
    }

    // 按照作业长短排序
    void _sort_l(PCB*& head, size_t curtime)
    {
        if (NULL == head || NULL == head->next)
            return;
        PCB* prev = head;
        PCB* cur = prev->next;
        int size = 0;  //计算进程的数目
        PCB* tmp = head;
        while (tmp)
        {
            ++size;
            tmp = tmp->next;
        }

        for (int i = 0; cur->arrtime < curtime && i < size - 1; i++)
        {
            if (prev->arrtime > curtime)
            {//作业还没到达就不排序
                return;
            }
```

```
            for (int j = 0; j < size - i - 1; j++)
            {
                if (cur && cur->arrtime <= curtime)
                {
                    int ptime = prev->sertime;
                    int ctime = cur->sertime;
                    if (ptime > ctime)
                    {
                        _Swap(prev, cur);
                    }
                }
                prev = cur;
                cur = cur->next;
            }
            prev = head;
            cur = prev->next;
        }
    }

    void _Swap(PCB * prev, PCB * cur)
    {
        swap(prev->arrtime, cur->arrtime);
        swap(prev->Num, cur->Num);
        swap(prev->sertime, cur->sertime);
    }

private:
    PCB *   _head;
    size_t  _curtime;
    size_t  _tasknum;   //作业个数
};
#endif
```

测试代码

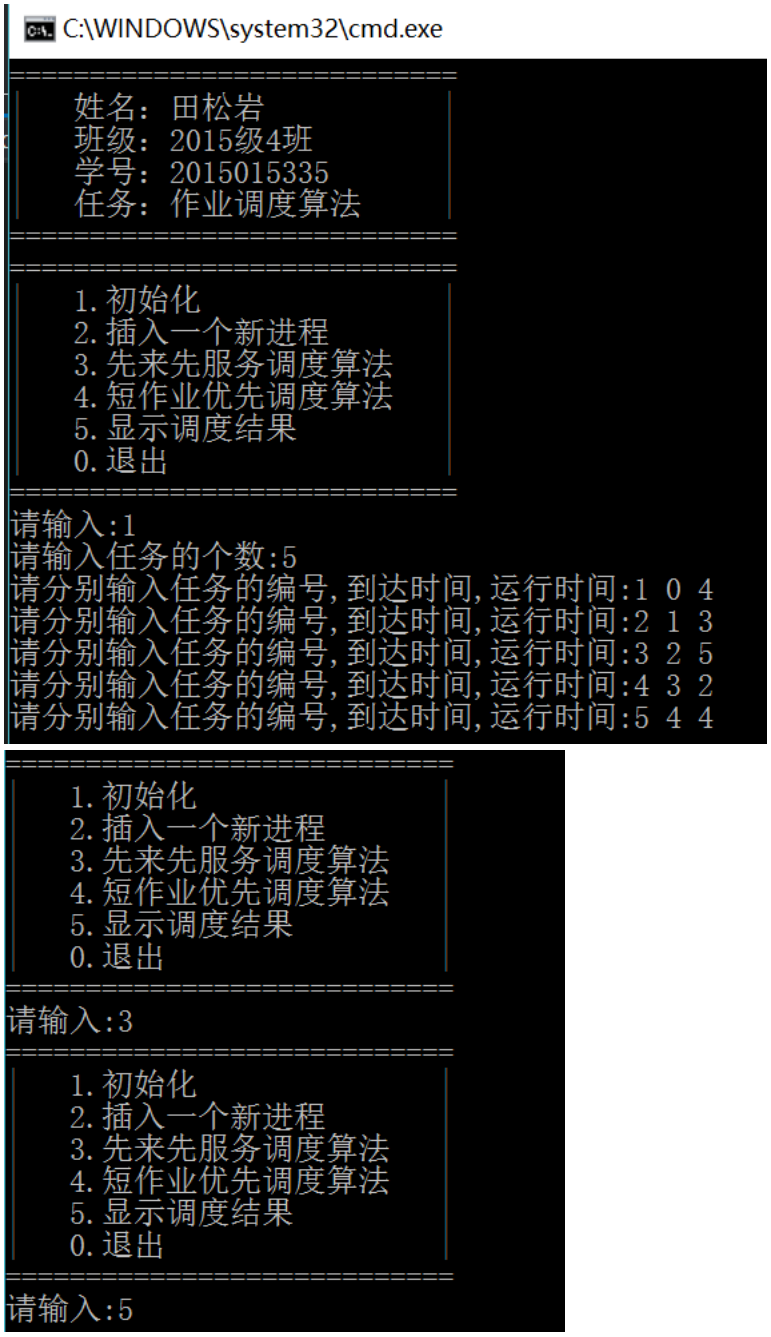

```
/*
调用头文件
实现作业调度过程
*/

#define _CRT_NOWANRINGS
#include"diaodu.h"//引入自定义的头文件
int main()
{
    cout << "============================" << endl;
    cout << "|   姓名：田松岩           |" << endl;
    cout << "|   班级：2015级4班        |" << endl;
    cout << "|   学号：2015015335       |" << endl;
    cout << "|   任务：作业调度算法     |" << endl;
    cout << "============================" << endl;
    int select = 1;//设默认的选择值
    scheduling mytask;//实例化一个调度类
    while (select)
    {
        cout << "============================" << endl;
        cout << "|   1.初始化               |" << endl;
        cout << "|   2.插入一个新进程       |" << endl;
        cout << "|   3.先来先服务调度算法   |" << endl;
        cout << "|   4.短作业优先调度算法   |" << endl;
        cout << "|   5.显示调度结果         |" << endl;
        cout << "|   0.退出                 |" << endl;
        cout << "============================" << endl;
        int item = 0;//
        cout << "请输入:";
        cin >> select;//选择要执行的命令
        switch (select)
        {
        case 1:
            mytask.Init_task();
            break;
        case 2:
            mytask.Push();
            break;
        case 3:
            mytask.FIFS();
```

```
            break;
        case 4:
            mytask.Short();
            break;
        case 5:
            mytask.Print();
            cout << endl;
            break;
        default:
            break;
        }
    }
    return 0;
}

//测试实例
/*
5
1 0 4
2 1 3
3 2 5
4 3 2
5 4 4
*/
```

二、程序运行截图


C:\WINDOWS\system32\cmd.exe
姓名：田松岩
班级：2015级4班
学号：2015015335
任务：作业调度算法
1.初始化
2.插入一个新进程
3.先来先服务调度算法
4.短作业优先调度算法
5.显示调度结果
0.退出
请输入:1
请输入任务的个数:5
请分别输入任务的编号,到达时间,运行时间:1 0 4
请分别输入任务的编号,到达时间,运行时间:2 1 3
请分别输入任务的编号,到达时间,运行时间:3 2 5
请分别输入任务的编号,到达时间,运行时间:4 3 2
请分别输入任务的编号,到达时间,运行时间:5 4 4
1.初始化
2.插入一个新进程
3.先来先服务调度算法
4.短作业优先调度算法
5.显示调度结果
0.退出
请输入:3
1.初始化
2.插入一个新进程
3.先来先服务调度算法
4.短作业优先调度算法
5.显示调度结果
0.退出
请输入:5

进程号	到达时间	服务时间	开始时间	完成时间	周转时间	带权周转时间
1	0	4	0	4	4	1.00
2	1	3	4	7	6	2.00
3	2	5	7	12	10	2.00
4	3	2	12	14	11	5.50
5	4	4	14	18	14	3.50

```
============================
|    1.初始化              |
|    2.插入一个新进程      |
|    3.先来先服务调度算法  |
|    4.短作业优先调度算法  |
|    5.显示调度结果        |
|    0.退出                |
============================
```

```
============================
请输入:4
============================
|    1.初始化              |
|    2.插入一个新进程      |
|    3.先来先服务调度算法  |
|    4.短作业优先调度算法  |
|    5.显示调度结果        |
|    0.退出                |
============================
请输入:5
```

进程号	到达时间	服务时间	开始时间	完成时间	周转时间	带权周转时间
1	0	4	0	4	4	1.00
4	3	2	4	6	3	1.50
2	1	3	6	9	8	2.67
5	4	4	9	13	9	2.25
3	2	5	13	18	16	3.20

三、两种调度算法的特点

先来先服务：

有利于长作业而不利于短作业

有利于 CPU 繁忙型作业而不利于 I/O 繁忙型作业

短作业优先调度

有效降低作业的平均等待时间，提高系统吞吐量

对长作业不利

该算法完全未考虑作业的紧迫程度，因而不能保证紧迫性作业会被及时处理

由于作业的长短含主观因素，不一定能真正做到短作业优先