

### 

### 实验报告

**课程名称:** **实用操作系统教程**

**设计课题: 作业调度算法的设计与实现**

**指导教师: 熊婷**

**专 业：** 计算机科学与技术

**班 级：** 1982066班

**学 号：** 198206606

**姓 名：** 陈文龙

**成 员：** 陈文龙

**二O 二 一 年 12 月 4 日**

**目 录**

[1.实验目的 1](#_Toc90209882)

[2.实验内容和要求 1](#_Toc90209883)

[3.实验原理 1](#_Toc90209884)

[4.作业调度算法 2](#_Toc90209885)

[5.源程序及结果 3](#_Toc90209886)

[**5.1 代码 3**](#_Toc90209887)

[**5.2 结果 10**](#_Toc90209888)

[6.总结 14](#_Toc90209892)

[参考文献： 14](#_Toc90209893)

# 1.实验目的

通过编写程序实现作业调度算法，使学生了解作业调度在操作系统中的作用，以加深对作业调度算法的理解。

# 2.实验内容和要求

自选一种语言编写一个在单道批处理系统中，分别按先来先服务调度、短作业优先调度算法、响应比高者优先（HRN）调度算法进行作业调度的模拟程序。

一、模拟数据的生成

1．允许用户指定作业的个数。

2．可以允许用户选择输入每个作业的到达时间和所需运行时间。

3．也可以允许用户选择通过伪随机数指定每个作业的到达时间和所需运行时间。

二、模拟程序的功能

1．按照模拟数据的到达时间和所需运行时间，执行FCFS, SJF和HRN调度算法，程序计算各作业的开始执行时间，各作业的完成时间，周转时间和带权周转时间，平均周转时间和平均带权周转时间。

2. 动态演示每调度一次，更新现在系统时刻，处于运行状态和等待各作业的相应信息（作业名、到达时间、所需的运行时间等）对于HRN算法，能在每次调度时显示各作业的响应比情况。

# 3.实验原理

假设在单道批处理环境下有四个作业JOB1、JOB2、JOB3、JOB4，已知它们进入系统的时间、估计运行时间。

1.先来先服务调度算法

2.短作业优先调度算法

3采用响应比高者优先（HRN）的调度算法:

作业 i 的等待时间=开始时间-到达时间

作业 i 的周转时间=运行结束时间-到达时间

作业i的带权周转时间 =周转时间/所需运行时间

响应比高者优先（HRN）：每次从后备队列中选择一个或若干个估计响应比最高的作业，将它们调入内存运行。

响应比Rp=作业响应时间/运行时间

=作业等待时间+作业运行时间

=1+作业等待时间/作业运行时间

**4.作业调度算法**

1) 采用先来先服务（FCFS）调度算法，即按作业到达的先后次序进行调度。总是首先调度在系统中等待时间最长的作业。

2) 短作业优先 (SJF) 调度算法，优先调度要求运行时间最短的作业。

3) 响应比高者优先(HRRN)调度算法，为每个作业设置一个优先权(响应比)，调度之前先计算各作业的优先权，优先数高者优先调度。RP (响应比)＝ 作业周转时间 / 作业运行时间=1+作业等待时间/作业运行时间

每个作业由一个作业控制块JCB表示，JCB可以包含以下信息：作业名、提交（到达）时间、所需的运行时间、所需的资源、作业状态、链指针等等。

作业的状态可以是等待W（Wait）、运行R（Run）和完成F（Finish）三种之一。每个作业的最初状态都是等待W。

# 5.源程序及结果

**5.1 代码：**

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#define getpch(type) (type \*)malloc(sizeof(type)) //为进程创建一个空间

struct worktime

{

    float Tb; //作业运行时刻

    float Tc; //作业完成时刻

    float Ti; //周转时间

    float Wi; //带权周转时间

};

struct jcb

{

    char name[10]; //作业名

    float subtime; //作业到达时间

    float runtime; //作业所需的运行时间

    char resource; //所需资源

    float Rp; //后备作业响应比

    char state; //作业状态

    int worked\_time; //已运行时间

    struct worktime wt;

    int need\_time; //要求运行时间

    int flag; //进程结束标志

    struct jcb \*link; //链指针

} \*ready = NULL, \*p;

typedef struct jcb JCB;

float T = 0;

int N;

JCB \*front, \*rear; //时间轮转法变量

void sort()

{

    JCB \*first, \*second;

    int insert = 0; //插入数

    if ((ready == NULL) || ((p->subtime) < (ready->subtime)))

    {

        p->link = ready;

        ready = p;

        T = p->subtime;

        p->Rp = 1;

    }

    else

    {

        first = ready;

        second = first->link;

        while (second != NULL)

        {

            if ((p->subtime) < (second->subtime))

            {

                p->link = second;

                first->link = p;

                second = NULL;

                insert = 1;

            }

            else

            {

                first = first->link;

                second = second->link;

            }

        }

        if (insert == 0)

            first->link = p;

    }

}

void SJFget()

{

    JCB \*front, \*mintime, \*rear;

    int ipmove = 0;

    mintime = ready;

    rear = mintime->link;

    while (rear != NULL)

    {

        if ((rear != NULL) && (T >= rear->subtime) && (mintime->runtime) > (rear->runtime))

        {

            front = mintime;

            mintime = rear;

            rear = rear->link;

            ipmove = 1;

        }

        else

            rear = rear->link;

    }

    if (ipmove == 1)

    {

        front->link = mintime->link;

        mintime->link = ready;

    }

    ready = mintime;

}

void HRNget()

{

    JCB \*front, \*mintime, \*rear;

    int ipmove = 0;

    mintime = ready;

    rear = mintime->link;

    while (rear != NULL)

        if ((rear != NULL) && (T >= rear->subtime) && (mintime->Rp) < (rear->Rp))

        {

            front = mintime;

            mintime = rear;

            rear = rear->link;

            ipmove = 1;

        }

        else

            rear = rear->link;

    if (ipmove == 1)

    {

        front->link = mintime->link;

        mintime->link = ready;

    }

    ready = mintime;

}

void creatJCB() //为每个作业创建一个JCB并初始化形成一个循环链队列

{

    JCB \*p, \*l;

    int i = 0;

    l = (JCB \*)malloc(sizeof(JCB));

    printf("\n 请输入作业的个数:");

    scanf("%d", &N);

    printf("\n 作业号No.%d:\n", i);

    printf("\n请输入作业的名字:");

    scanf("%s", l->name);

    printf("\n请输入作业的时间:");

    scanf("%d", &l->need\_time);

    l->state = 'r'; //作业初始状态为就绪

    l->worked\_time = 0;

    l->link = NULL;

    l->flag = 0;

    front = l;

    for (i = 1; i < N; i++)

    {

        p = (JCB \*)malloc(sizeof(JCB));

        printf("\n 作业号No.%d:\n", i);

        printf("\n请输入作业的名字:");

        scanf("%s", p->name);

        printf("\n请输入作业的时间:");

        scanf("%d", &p->need\_time);

        p->state = 'r';

        p->worked\_time = 0;

        p->flag = 0;

        l->link = p;

        l = l->link;

    }

    rear = l;

    rear->link = front;

}

void output() //进程输出函数

{

    int j;

    printf("name runtime needtime state\n");

    for (j = 1; j <= N; j++)

    {

        printf(" %-4s\t%-4d\t%-4d\t%-c\n", front->name, front->worked\_time, front->need\_time, front->state);

        front = front->link;

    }

    printf("\n");

}

int judge(JCB \*p) //判断所有进程运行结束

{

    int flag = 1, i;

    for (i = 0; i < N; i++)

    {

        if (p->state != 'e')

        {

            flag = 0;

            break;

        }

        p = p->link;

    }

    return flag;

}

void RRget() //时间片轮转算法

{

    JCB \*s;

    int flag1 = 0;

    s = (JCB \*)malloc(sizeof(JCB));

    s = front;

    printf("\n--------------------------------------------\n");

    output();

    printf("请输入任意一键继续\n");

    getch(); //按任意键继续

    s = front;

    while (flag1 != 1)

    {

        if (s->state == 'r')

        {

            s->worked\_time++;

            s->need\_time--;

            if (s->need\_time == 0)

                s->state = 'e';

            output();

            printf("请输入任意一键继续...\n");

            getch();

        }

        if (s->state == 'e' && s->flag == 0)

        {

            printf("进程%s已经运行完成！\n\n", s->name);

            s->flag = 1;

        }

        s = s->link;

        flag1 = judge(s);

    }

    printf("--------------------------------------------\n");

}

void input()

{

    int i, num;

    printf("\n 请输入作业的个数:");

    scanf("%d", &num);

    for (i = 0; i < num; i++)

    {

        printf("\n 作业号No.%d:\n", i);

        p = getpch(JCB);

        printf("\n 输入作业名:");

        scanf("%s", p->name);

        printf("\n 输入作业到达时刻:");

        scanf("%f", &p->subtime);

        printf("\n 输入作业运行时间:");

        scanf("%f", &p->runtime);

        printf("\n");

        p->state = 'w';

        p->link = NULL;

        sort();

    }

}

int space()

{

    int l = 0;

    JCB \*jr = ready;

    while (jr != NULL)

    {

        l++;

        jr = jr->link;

    }

    return (l);

}

void disp(JCB \*jr, int select)

{

    if (select == 3)

        printf("\n 作业  到达时间   服务时间  响应比  运行时刻  完成时刻  周转时间   带权周转时间 \n");

    else

        printf("\n 作业  到达时间   服务时间  运行时刻  完成时刻  周转时间   带权周转时间 \n");

    printf(" |%s\t", jr->name);

    printf(" |%.2f\t  ", jr->subtime);

    printf(" |%.2f\t", jr->runtime);

    if (select == 3 && p == jr)

        printf("|%.2f   ", jr->Rp);

    if (p == jr)

    {

        printf("|%.2f\t ", jr->wt.Tb);

        printf(" |%.2f  ", jr->wt.Tc);

        printf("  |%.2f\t", jr->wt.Ti);

        printf("  |%.2f", jr->wt.Wi);

    }

    printf("\n");

}

int destroy()

{

    printf("\n 作业 [%s] 已完成.\n", p->name);

    free(p);

    return (1);

}

void check(int select)

{

    JCB \*jr;

    printf("\n \*\*\*\* 当前正在运行的作业是:%s", p->name);

    disp(p, select);

    jr = ready;

    printf("\n \*\*\*\*当前就绪队列状态为:\n");

    while (jr != NULL)

    {

        jr->Rp = (jr->runtime + T - jr->subtime) / jr->runtime;

        disp(jr, select);

        jr = jr->link;

    }

    destroy();

}

void running(JCB \*jr)

{

    if (T >= jr->subtime)

        jr->wt.Tb = T;

    else

        jr->wt.Tb = jr->subtime;

    jr->wt.Tc = jr->wt.Tb + jr->runtime;

    jr->wt.Ti = jr->wt.Tc - jr->subtime;

    jr->wt.Wi = jr->wt.Ti / jr->runtime;

    T = jr->wt.Tc;

}

int main()

{

    int select = 0, len, h = 0;

    float sumTi = 0, sumWi = 0;

    printf("\t---\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*---\n");

    printf("请选择作业调度算法的方式:\n");

    printf("\t1.FCFS 2.SJF 3.HRN 4.RR\n\n");

    printf("\t---\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*---\n");

    printf("请输入作业调度算法序号(1-4):");

    scanf("%d", &select);

    if (select == 4)

    {

        creatJCB();

        RRget();

    }

    else

    {

        input();

        len = space();

        while ((len != 0) && (ready != NULL))

        {

            h++;

            printf("\n 执行第%d个作业 \n", h);

            p = ready;

            ready = p->link;

            p->link = NULL;

            p->state = 'R';

            running(p);

            sumTi += p->wt.Ti;

            sumWi += p->wt.Wi;

            check(select);

            if (select == 2 && h < len - 1)

                SJFget();

            if (select == 3 && h < len - 1)

                HRNget();

            printf("\n 按任意一键继续......");

            getchar();

            getchar();

        }

        printf("\n\n 作业已经完成.\n");

        printf("\t 此组作业的平均周转时间：%.2f\n", sumTi / h);

        printf("\t 此组作业的带权平均周转时间：%.2f\n", sumWi / h);

        getchar();

    }

}

## **5.2结果：**

设有3个作业，分别为p0,p1,p2；

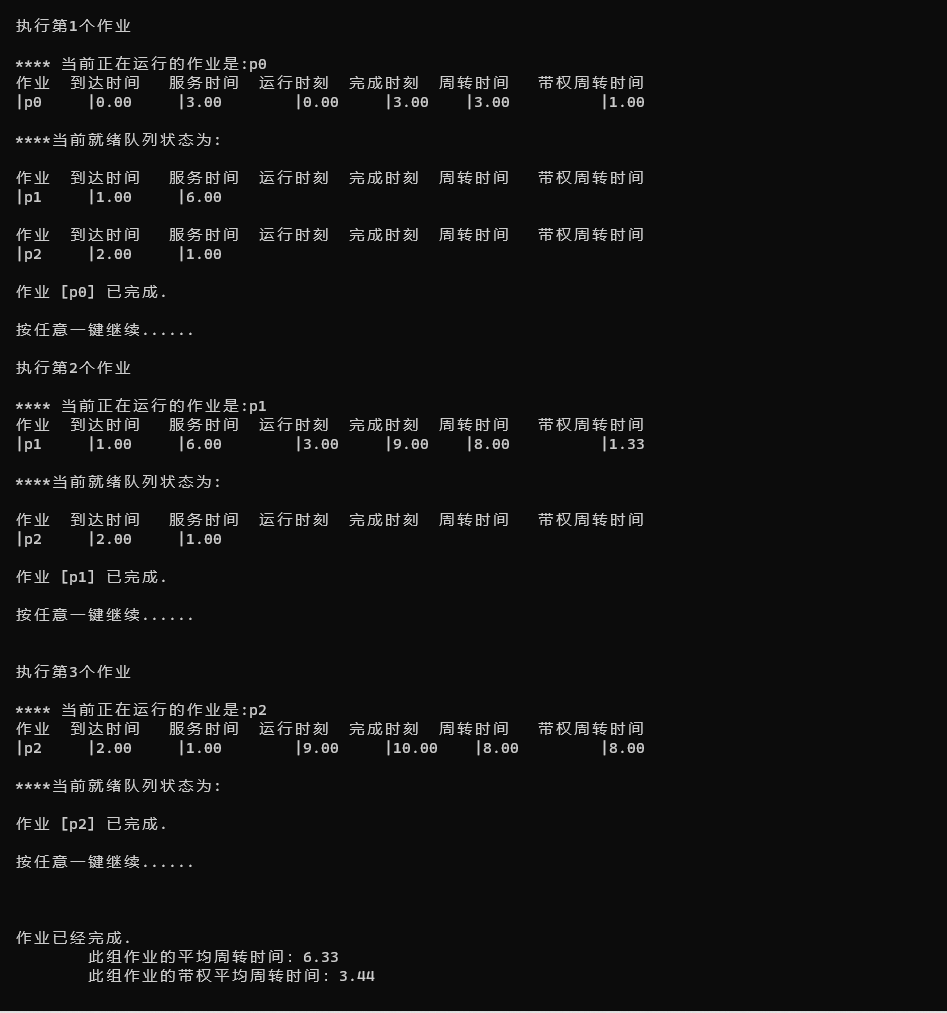
p0到达时间为0，运行时间为3；

p1到达时间为1，运行时间为6；

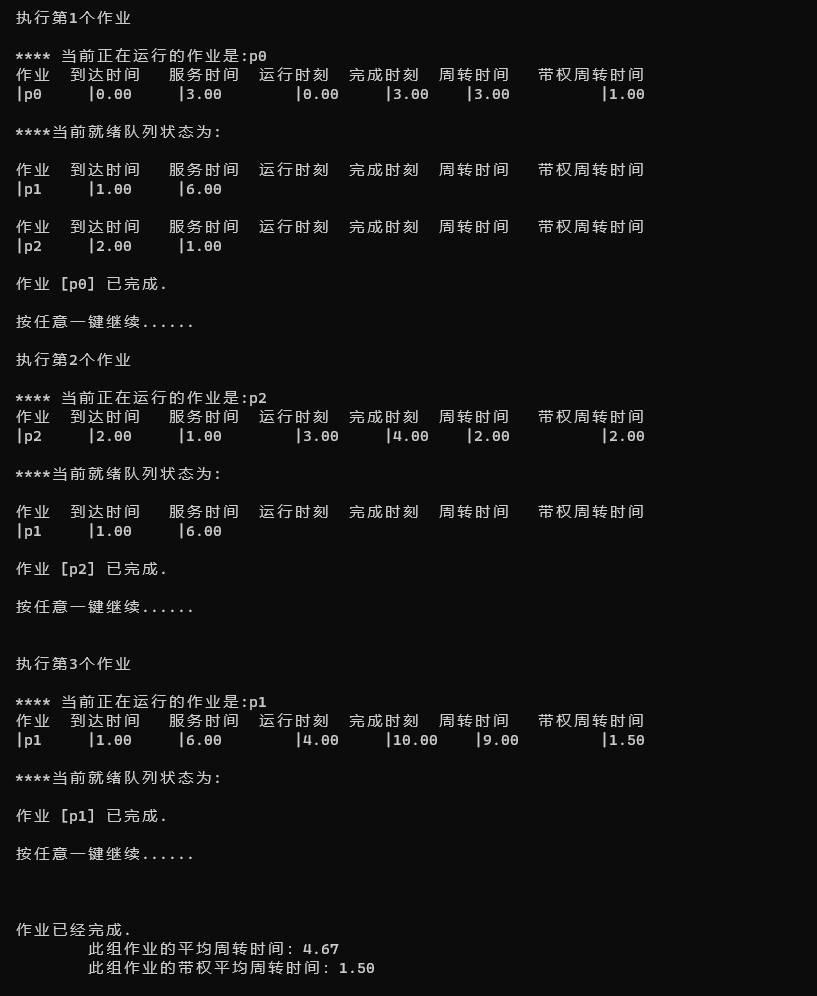
p2到达时间为2，运行时间为1；

那么在不同调度算法下的运行结果如下：

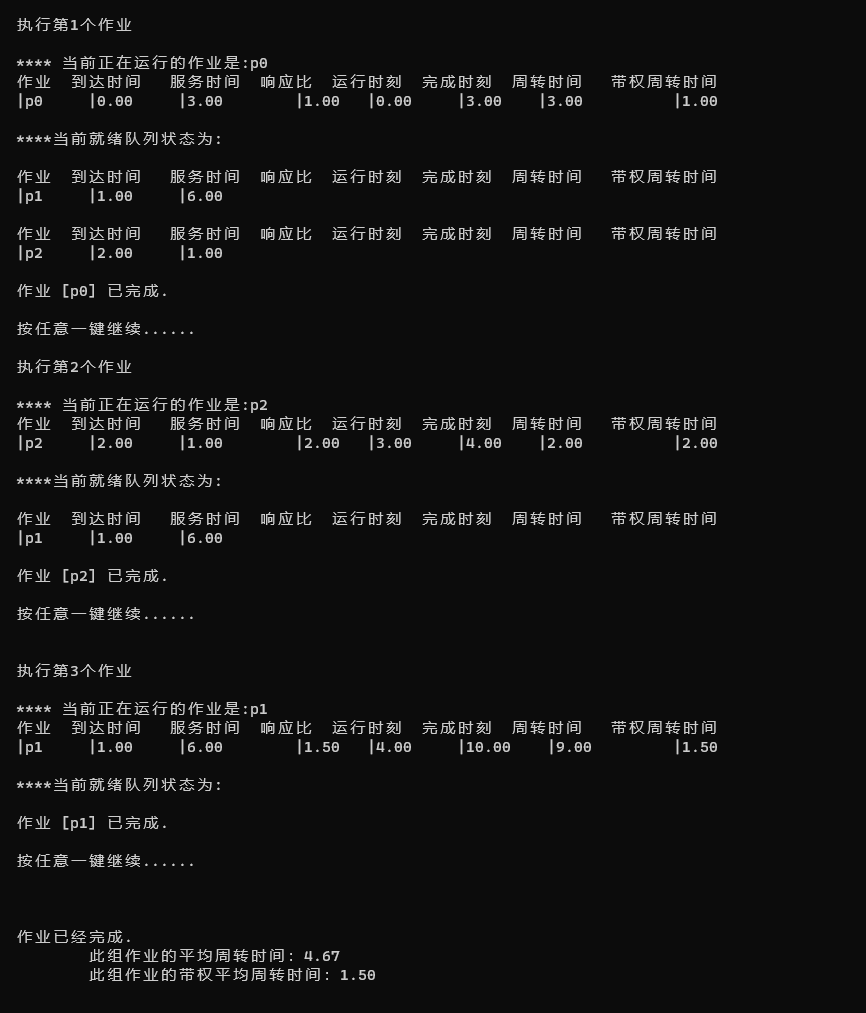
### FCFS（先来先服务调度算法）



### SJF（短作业优先调度算法）



### HRN（高响应比优先调度算法）



# 6.总结

通过本次实验了解到算法很重要，又更加明白算法本身可以节约时间，而且不同的函数之间在调用的时候要注意很多的问题。开始时对作业调度不是很那么地清晰，通过做这个实验，现在对操作系统的作业调度有了清晰的理解，感觉在这个过程中自己就是充当了调度员的角色。

参考文献：

［1］李建伟主编. 实用操作系统教程[M]. 北京：清华大学出版社，2016