一、实验目的与要求：

实验目的

1. 了解操作系统对 CPU 存在三级调度
2. 掌握作业调度和进程调度的实现原理
3. 掌握常用的几种调度算法
4. 掌握死锁问题产生的必要条件以及预防和避免措施

实验内容

1. 高优先权优先调度和基于时间片的轮转调度算法
2. 常用的几种实时调度算法
3. 多处理机环境下的进程调度方式
4. 死锁的基本概念，预防死锁的方法和银行家算法

实验环境

硬件：桌面 PC

软件：Linux 操作系统

二、方法、步骤：(说明程序相关的算法原理或知识内容，程序设计的思路和方法，可以用流程图表述，程序主要数据结构的设计、主要函数之间的调用关系等)

实验要求：

- 模拟在单处理器多进程操作系统的 CPU 调度。主要实现各种调度算法。
- 进程 PCB 结构：标识符、进程名称、到达时间、服务时间、剩余运行时间、已使用时间、进程状态。其中进程状态有三种：就绪 R，运行 E，结束 F。
- 剩余运行时间和已使用时间两个属性用于时间片轮转算法和多级反馈队列算法。进程每使用完一个时间片，已使用时间就会增加一个时间片的长度.其中，剩余运行时间=服务时间 - 已使用时间。
- 关于测试数据：一种是程序内置数据，通过语句已经预先设置好，存放在数组 pcbdata 中，数据来源于课本 P101 第二个表格的数据，共 5 个进程。另一种方法是手工输入，使用函数 input()，输入数据同样存放在数组 pcbdata 中。如果使用 input 函数，内置数据将不会使用。

1、阅读理解例程，掌握例程的运作流程。运行例程，理解**先来先服务算法**的调度原理和运行结果。

2、参考先来先服务算法，实现四种调度算法：**短作业优先、高响应比、时间片轮转、多级反馈队列**。（多级反馈队列为**选做。**）（除了多级反馈队列，其他算法采用**非抢占调度**）

a) 短作业优先算法使用**例题一数据或程序内置数据**，要求运行结果给出调度顺序、完成时间、周转时间、带权周转时间

b) 高响应比算法使用**例题二的数据**，要求运行结果给出调度顺序、完成时间、周转时间、带权周转时间

c) 时间片轮转算法可以使用**程序内置数据**，要求运行结果给出每个时间片是被哪个进程使用，每个进程完成时，要修改状态并输出提示。

d) 多级反馈队列算法使用**例题三的数据**，要求运行结果给出正确的进程调度顺序和过程描述。

e) 为了检验算法的正确性，大家对测试数据用**笔算推导**，然后与程序运

行结果对比，检查算法是否正确。

例题一：在单处理机环境下，对 4 个作业 Job1、Job2、Job3、Job4 进行非抢占式调度，它们的到达时间均为 1，所需运行时间分别为 9、16、3、11。

例题二：在单处理机环境下，对 4 个进程 P1、P2、P3、P 4 进行非抢占式调度，它们的到达时间分别为 10、12、14、16，运行时间分别为 8、12、4、6。

例题三：在某一操作系统中对进程调度采用多级反馈队列调度算法。现设定采用三级反馈队列调度算法，三个队列分别为 I、II、III，对应时间片为 2、4、8。现有四个进程 A、B、C、D，到达时刻分别为 0、5、7、12，执行时间分别为 7、4、13、9。请写出整个进程调度过程。

目录：

每个算法包含如下部分：

- **数据**
- **算法流程图**
- **核心代码**
- **运行结果**
- **笔算推导**
- **算法分析**

三. 实验过程及内容：（对程序代码进行说明和分析，越详细越好，代码排版要整齐，可读性要高）

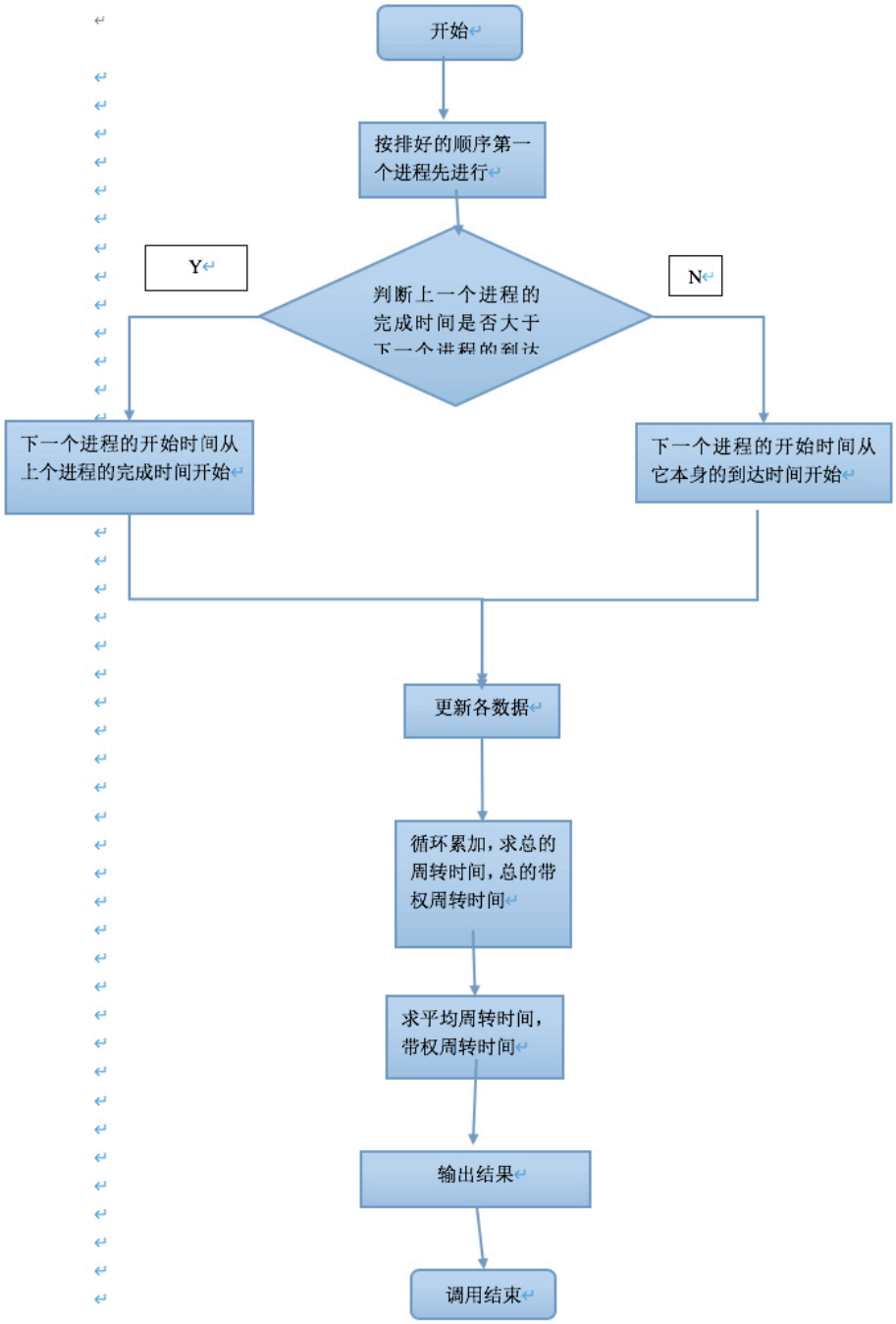
1.先来先服务算法 FCFS

- **数据：**使用程序内置数据 pcbdata

```
1.  PCB pcbdata[10]={
2.      //例程内置数据
```

```
    {1000,"A",0,4,4,0,'R'},
    {1001,"B",1,3,3,0,'R'},
    {1002,"C",2,5,5,0,'R'},
    {1003,"D",3,2,2,0,'R'},
    {1004,"E",4,4,4,0,'R'},
};
```

算法流程图：



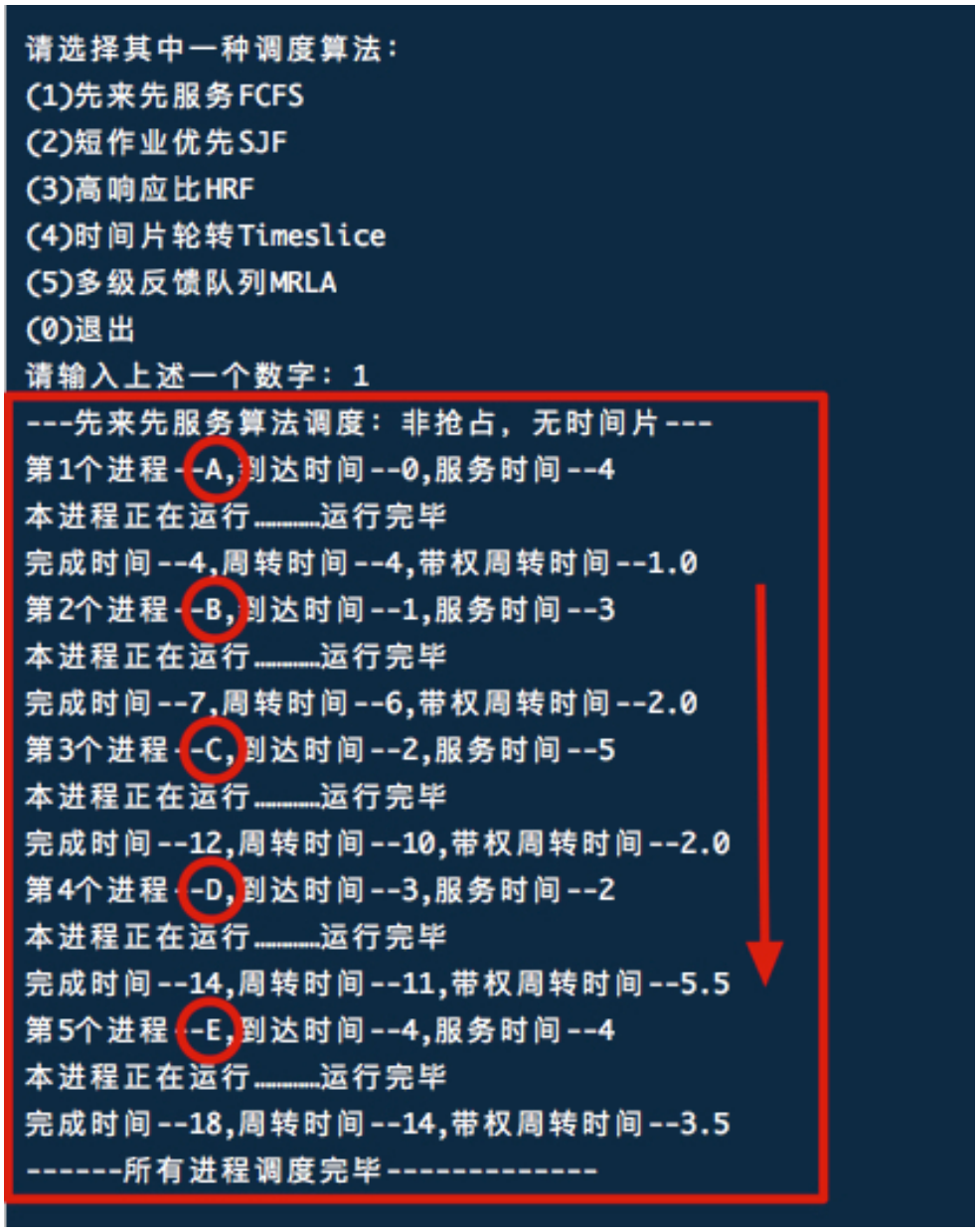
核心代码：

以到达时间排序，按从小到大顺序存在就绪队列（ready 数组），处于队

列头的进程先运行，依次从就绪队列取出进程直到队列为空。

```
void FCFS()
{
    //以到达时间为排序对象，从小到大排序，静态排序
    int i,j,temp;
    double k;
    for(i=0;i<num;i++)
    {order[i]=pcbdata[i].time_start;
    ready[i]=i;
    }
    for(i=0;i<num;i++)          //按到达时间排序
        for(j=i+1;j<num;j++)
        {
            if(order[i]>order[j])
            {
                temp=order[i];
                order[i]=order[j];
                order[j]=temp;
                temp=ready[i];
                ready[i]=ready[j];
                ready[j]=temp;
            }
        }
        printf("---先来先服务算法调度：非抢占，无时间片---\n");
        temp=pcbdata[ready[0]].time_start;
        for(i=0;i<num;i++)
        {
            printf("第%d 个进程--%s,",i+1,pcbdata[ready[i]].name);
            printf("到达时间--%d,服务时间
--%d\n",pcbdata[ready[i]].time_start,pcbdata[ready[i]].time_need);
            printf("本进程正在运行……");
            _sleep(1);
            printf("运行完毕\n");
            temp+=pcbdata[ready[i]].time_need;  //完成时间
            j=temp-pcbdata[ready[i]].time_start;  //周转时间
            k=(float)j/pcbdata[ready[i]].time_need;  //带权周转时间
            printf("完成时间--%d,周转时间--%d,带权周转时间
--%.1f\n",temp,j,k);
        }
        printf("------所有进程调度完毕-------------\n");
}
```

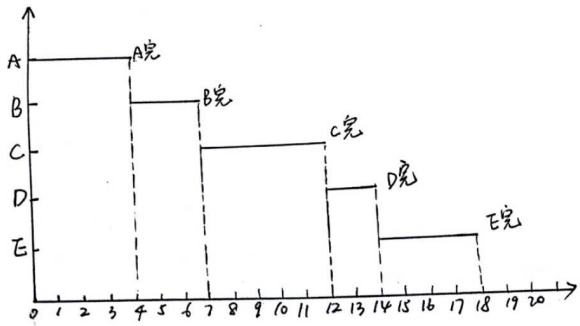
- 运行结果：


请选择其中一种调度算法：
(1)先来先服务FCFS
(2)短作业优先SJF
(3)高响应比HRF
(4)时间片轮转Timeslice
(5)多级反馈队列MRLA
(0)退出
请输入上述一个数字：1
---先来先服务算法调度：非抢占，无时间片---
第1个进程--A,到达时间--0,服务时间--4
本进程正在运行.....运行完毕
完成时间--4,周转时间--4,带权周转时间--1.0
第2个进程--B,到达时间--1,服务时间--3
本进程正在运行.....运行完毕
完成时间--7,周转时间--6,带权周转时间--2.0
第3个进程--C,到达时间--2,服务时间--5
本进程正在运行.....运行完毕
完成时间--12,周转时间--10,带权周转时间--2.0
第4个进程--D,到达时间--3,服务时间--2
本进程正在运行.....运行完毕
完成时间--14,周转时间--11,带权周转时间--5.5
第5个进程--E,到达时间--4,服务时间--4
本进程正在运行.....运行完毕
完成时间--18,周转时间--14,带权周转时间--3.5
------所有进程调度完毕----------

- 笔算推导：

1. 先来先服务算法

作业	到达时间	运行时间	开始时刻	完成时刻	周转时间	带权周转时间
A	0	4	0	4	4	1
B	1	3	4	7	6	2
C	2	5	7	12	10	2
D	3	2	12	14	11	5.5
E	4	4	14	18	14	3.5



- **算法分析：**

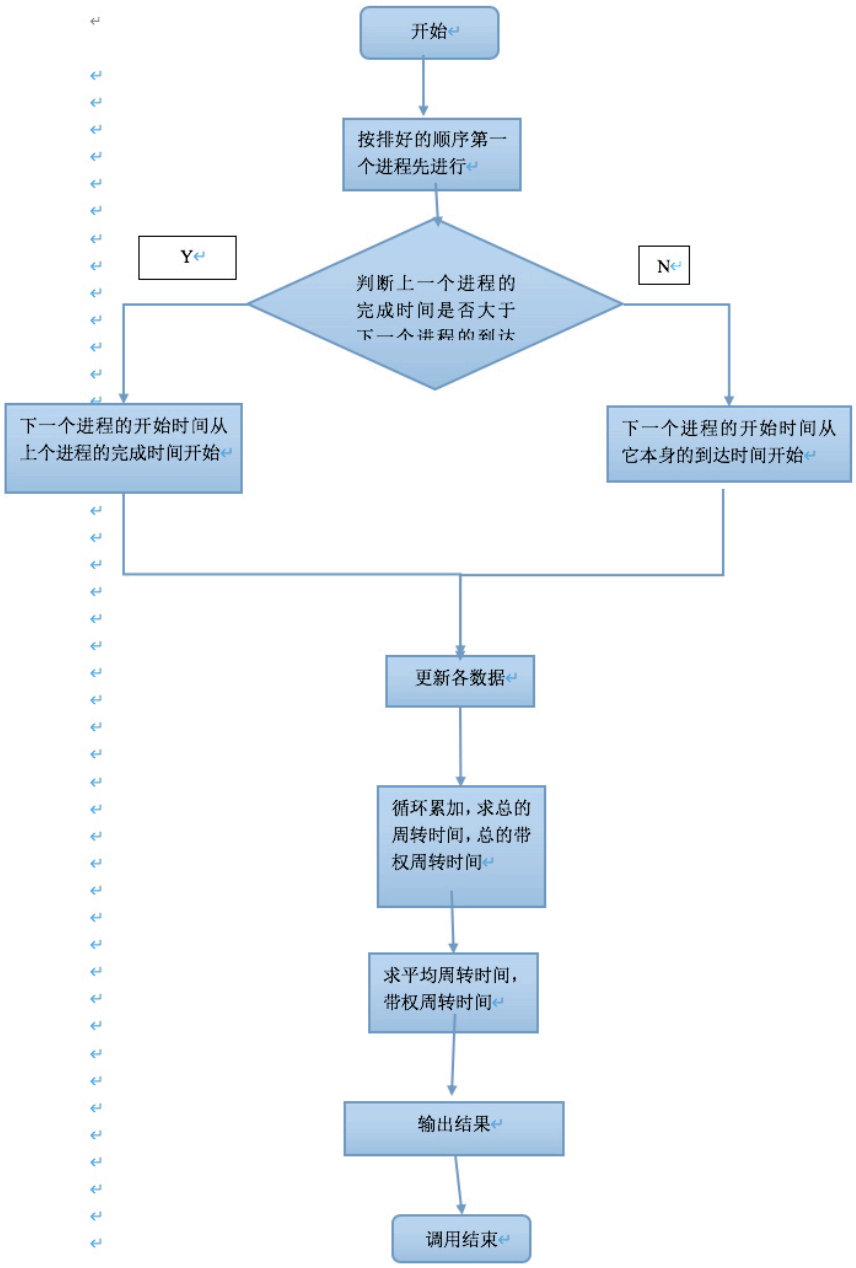
  优点：实现简单，算法开销小，有利于长时间作业（进程）
  缺点：不利于短时间作业（进程）

2. 短作业优先算法 SJF：

- **数据：**例题一数据 pcbdata1

```
PCB pcbdata1[10]={
    //例题一数据
    {1000,"Job1",1,9,9,0,'R'},
    {1001,"Job2",1,16,16,0,'R'},
    {1002,"Job3",1,3,3,0,'R'},
    {1003,"Job4",1,11,11,0,'R'},
};
```

- **算法流程图：**



核心代码：

```
1.          printf("---短作业算法调度：非抢占，无时间片---\n");
2.          int t_ready[10];        //就绪队列，存放进程在 pcbdata 中的位置
3.
4.          int t_order[10];        //记录排序使用哪个数值作为排序对象
5.          for(i=0;i<num1;i++)
6.          {
7.              t_order[i]=pcbdata1[ready[i]].time_need;//服务时间作为排序对象
8.              t_ready[i]=ready[i];
```

```
        }
        time=order[0];
        for(l=0;l<num1;l++){
            //判断到达的进程数，用 temp_num 存放
            for(i=0;i<num&&pcbdata1[ready[i]].time_start<=time;i++)
                temp_num=i+1;
            //把到达的进程按服务时间大小进行排序
            for(i=0;i<temp_num;i++)
                for(j=i+1;j<temp_num;j++)
                {
                    if(t_order[i]>t_order[j]&&t_order[j]!=0||t_order[i]==0
)
                    {
                        temp=t_order[i];
                        t_order[i]=t_order[j];
                        t_order[j]=temp;
                        temp=t_ready[i];
                        t_ready[i]=t_ready[j];
                        t_ready[j]=temp;
                    }
                }

                printf("第%d 个进程
--%s,",l+1,pcbdata1[t_ready[0]].name);
                printf("正在运行……");
                _sleep(1);
                printf("运行完毕\n");
                time+=pcbdata1[t_ready[0]].time_need;
                j=time-pcbdata1[t_ready[0]].time_start;
                k=(float)j/pcbdata1[t_ready[0]].time_need;
                t_order[0]=0;
                printf("完成时间--%d,周转时间--%d,带权周转时间
--%.1f\n",time,j,k);
```

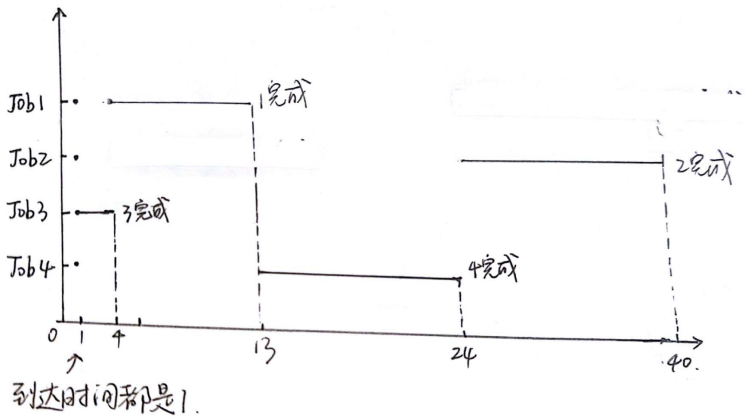
运行结果：

```
请选择其中一种调度算法：
(1)先来先服务FCFS
(2)短作业优先SJF
(3)高响应比HRF
(4)时间片轮转Timeslice
(5)多级反馈队列MRLA
(0)退出
请输入上述一个数字：2
---短作业算法调度：非抢占，无时间片---
第1个进程--Job3,正在运行.....运行完毕
完成时间--4,周转时间--3,带权周转时间--1.0
第2个进程--Job1,正在运行.....运行完毕
完成时间--13,周转时间--12,带权周转时间--1.3
第3个进程--Job4,正在运行.....运行完毕
完成时间--24,周转时间--23,带权周转时间--2.1
第4个进程--Job2,正在运行.....运行完毕
完成时间--40,周转时间--39,带权周转时间--2.4
------所有进程调度完毕----------
```

- **笔算推导：**

2.短作业优先调度算法 SJF（非抢占式）

进程名	Job1	Job2	Job3	Job4
到达时间	1	1	1	1
运行时间	9	16	3	11
完成时间	13	40	4	24
周转时间	12	39	3	23
带权周转时间	1.3	2.4	1	2.1



- **算法分析：**

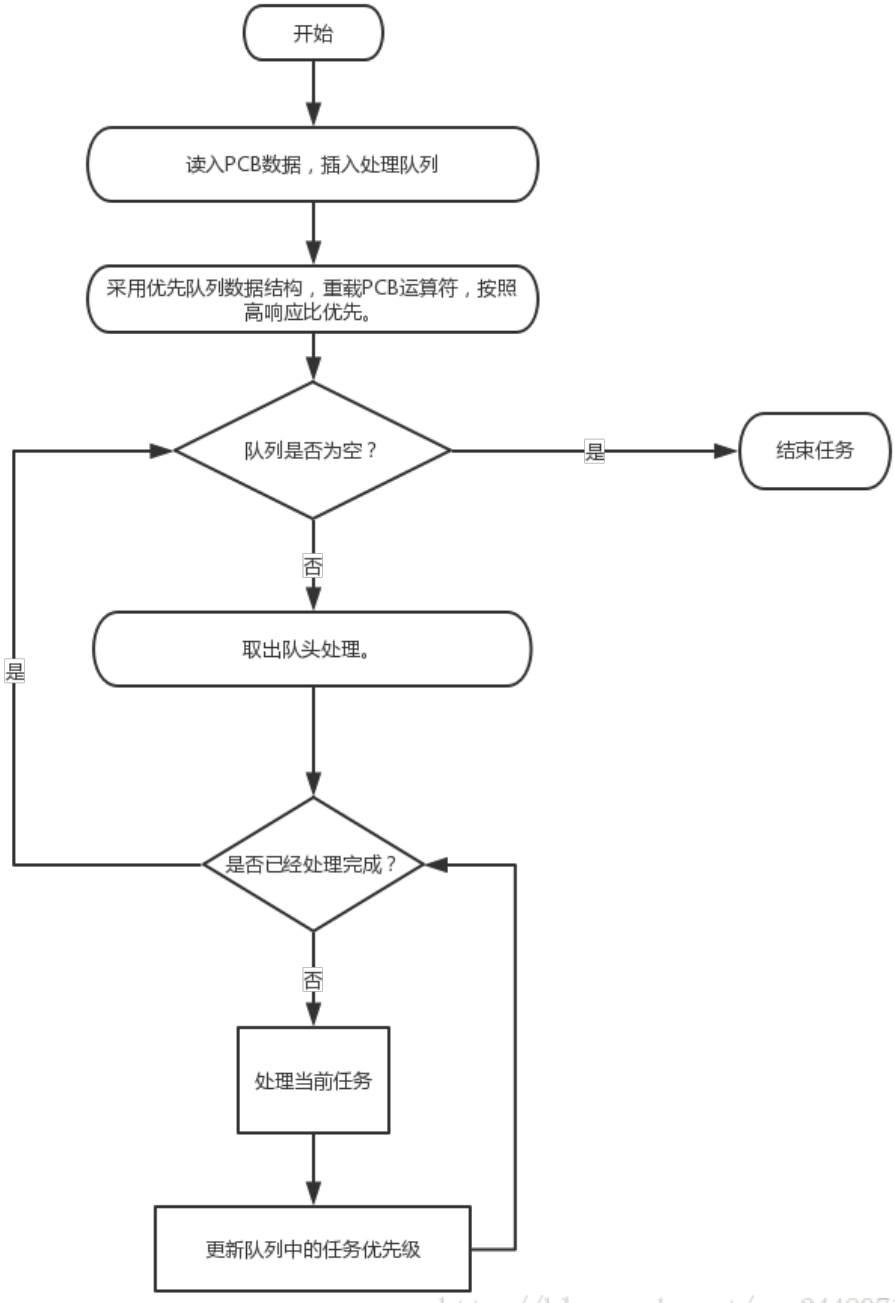
  优点：有利于短作业或短进程
  缺点：导致长作业或进程长时间不被调度，不能保证实时性，执行时间一般基于用户估算，准确性不足

3. 高响应比算法 HRF：

- **数据：**例题二数据 pcbdata2

```
PCB pcbdata2[10]={
    //例题二数据
    {1000,"P1",10,8,8,0,'R'},
    {1001,"P2",12,12,12,0,'R'},
    {1002,"P3",14,4,4,0,'R'},
    {1003,"P4",16,6,6,0,'R'},
};
```

- **算法流程图**



核心代码：

```
1.  printf("---高响应比算法调度：非抢占，无时间片---\n");
2.          int t_ready[10];
3.          int t_order[10];
4.          for(i=0;i<num2;i++)
5.          {
6.              t_order[i]=1;
7.              t_ready[i]=ready[i];
8.          }
9.          time=order[0];
10.         for(l=0;l<num2;l++){
```

```
            //判断到达进程数
            for(i=0;i<num && pcbdata2[ready[i]].time_start <= time;i++)
                temp_num=i+1;
            for(i=0;i<temp_num;i++)   //计算已到达进程的优先权
            {
                if(t_order[i])
t_order[i]=(time-pcbdata2[t_ready[i]].time_start+ pcbdata2[t_ready[i]].time_need)/pcbdata2[t_ready[i]].time_need;
            }
            for(i=0;i<temp_num;i++)    //按优先权排序
                for(j=i+1;j<temp_num;j++)
                {
                    if(t_order[i]<t_order[j])
                    {
                        temp=t_order[i];
                        t_order[i]=t_order[j];
                        t_order[j]=temp;
                        temp=t_ready[i];
                        t_ready[i]=t_ready[j];
                        t_ready[j]=temp;
                    }
                }
                printf("第%d 个进程--%s,",l+1,pcbdata2[t_ready[0]].name);
                printf("正在运行……");
                _sleep(1);
                printf("运行完毕\n");
                time+=pcbdata2[t_ready[0]].time_need;
                j=time-pcbdata2[t_ready[0]].time_start;
                k=(float)j/pcbdata2[t_ready[0]].time_need;
                t_order[0]=0;
                printf("完成时间--%d,周转时间--%d,带权周转时间--%.1f\n",time,j,k);
        }
        printf("------所有进程调度完毕-------------\n");
}
```

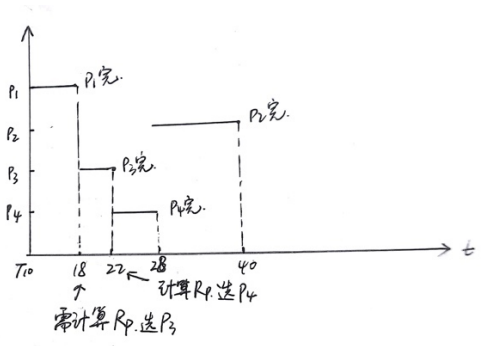
运行结果：

```
请选择其中一种调度算法：
(1)先来先服务FCFS
(2)短作业优先SJF
(3)高响应比HRF
(4)时间片轮转Timeslice
(5)多级反馈队列MRLA
(0)退出
请输入上述一个数字：3
---高响应比算法调度：非抢占，无时间片---
第1个进程--P1,正在运行.....运行完毕
完成时间--18,周转时间--8,带权周转时间--1.0
第2个进程--P3,正在运行.....运行完毕
完成时间--22,周转时间--8,带权周转时间--2.0
第3个进程--P4,正在运行.....运行完毕
完成时间--28,周转时间--12,带权周转时间--2.0
第4个进程--P2,正在运行.....运行完毕
完成时间--40,周转时间--28,带权周转时间--2.3
------所有进程调度完毕------------
```

- **笔算推导：**

3.高响应比算法HRF（非抢占）

作业	进入时刻	运行时间	开始时刻	完成时刻	周转时间	带权周转时间
P_1	10	8	10	18	8	1
P_2	12	12	28	40	28	2.3
P_3	14	4	18	22	8	2
P_4	16	6	22	28	12	2



		P_1	P_2	P_3	P_4
	T_s	8	12	4	6
时刻	T_{18}	\	$\frac{6}{12}$	$\frac{4}{4}$	$\frac{2}{6}$
	T_{22}	\	$\frac{10}{12}$	\	$\frac{8}{6}$
	T_{28}		0		

$R_P = 1 + \frac{T_W}{T_S}$ → 只需比较这一项.

其中，T_S为运行时间

T_W为等待时间（开始-进入时刻）

- **算法分析：**

  优点：对于等待时间相同的时候，服务时间愈短则优先权愈高，算法有利于短作业;对于服务时间相同的时候，等待时间愈长其优先权愈高，算法实现的是先来先服务;对于长作业，作业的优先级可以随等待时间的增加而提高，当其等待时间足够长时，其优先级便可升到很高保证能获得处理机。

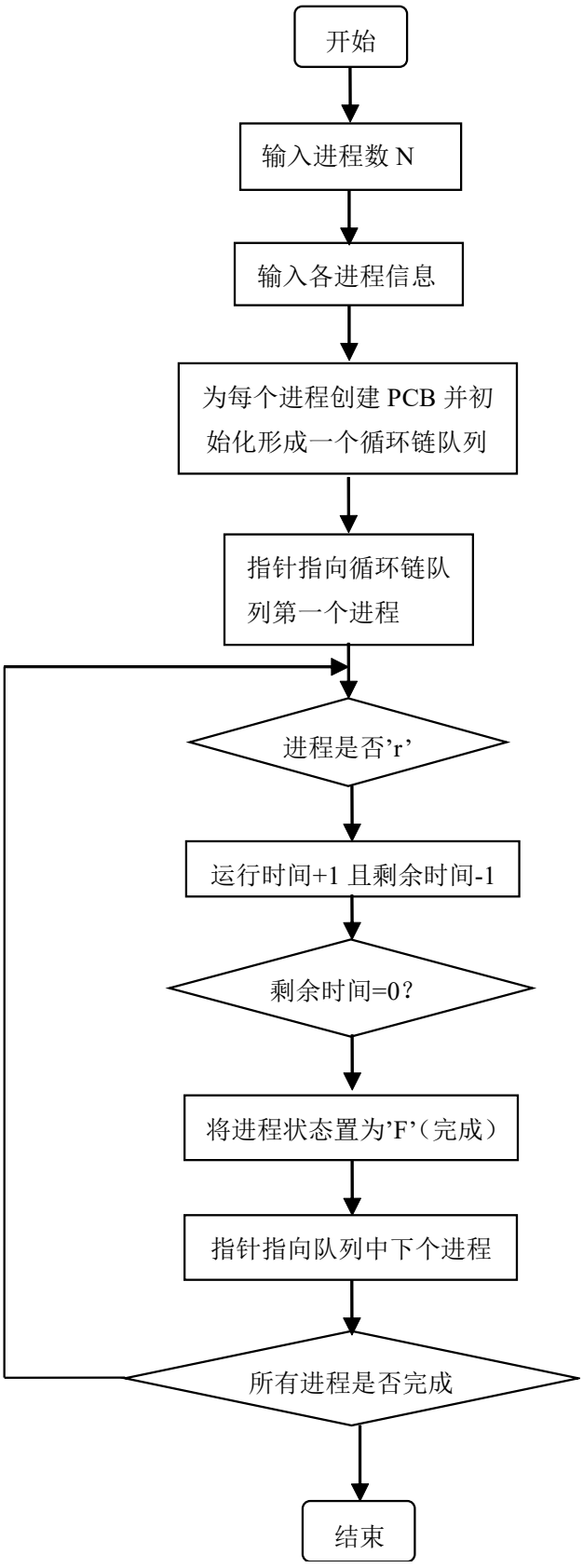
  缺点：每次调度前都要计算优先权，增加系统开销。

4.时间片轮转算法 Timeslice:

- **数据：**程序内置数据 pcbdata

```
1.  PCB pcbdata[10]={
2.      //例程内置数据
3.      {1000,"A",0,4,4,0,'R'},
4.      {1001,"B",1,3,3,0,'R'},
5.      {1002,"C",2,5,5,0,'R'},
6.      {1003,"D",3,2,2,0,'R'},
7.      {1004,"E",4,4,4,0,'R'},
8.  };
```

- **算法流程图**



代码分析：

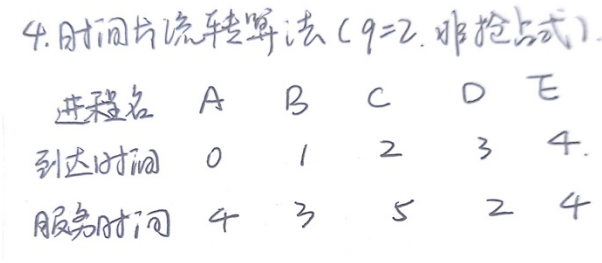
```
1.  printf("---时间片轮转算法调度：非抢占，时间片大小为 2---\n");
```

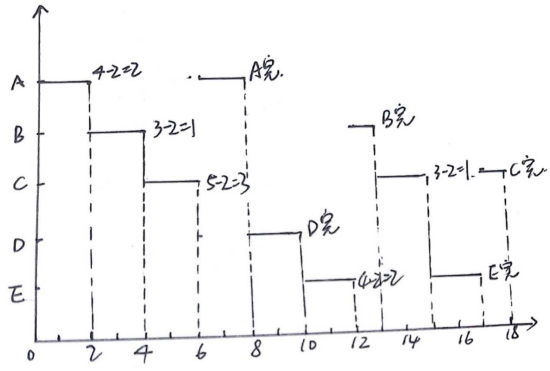
```
        int t_ready[10];
        for(i=0;i<num;i++)
        {
            t_ready[i]=ready[i];
        }
        time=order[0];
        for(l=0;done<num;l++){
        //判断到达的进程数
            for(i=0;i<num&&pcbdata[ready[i]].time_start<=time;i++)
                temp_num=i+1;

            if(time!=order[0]){
    //将已使用时间片的进程，即第一个移到队列末尾
                for(i=1;i<temp_num;i++){
                    temp=t_ready[i];
                    t_ready[i]=t_ready[i-1];
                    t_ready[i-1]=temp;
                }
            }
            if(pcbdata[t_ready[0]].state!='F'){
                printf("第%d 个时间片被进程%s 使
用,",l+1,pcbdata[t_ready[0]].name);
                printf("正在运行……\n    ");
                _sleep(1);
                printf("时间片使用完，所需时
间%d,",pcbdata[t_ready[0]].time_left);
                time+=2;
                pcbdata[t_ready[0]].time_used+=2;
                pcbdata[t_ready[0]].time_left-=2;
                printf("使用时间%d,还需时
间%d,",2,pcbdata[t_ready[0]].time_left);
    //判断进程是否结束
                if(pcbdata[t_ready[0]].time_left<=0){
                    printf("进程%s 结束\n",pcbdata[t_ready[0]].name);
                    done++;
                    pcbdata[t_ready[0]].state='F';
                }
                else
                    printf("进程%s 就绪\n",pcbdata[t_ready[0]].name);
            }
        }
        printf("------所有进程调度完毕-------------\n");
}
```

- 运行结果：

```
(4)时间片轮转Timeslice
(5)多级反馈队列MRLA
(0)退出
请输入上述一个数字：4
---时间片轮转算法调度：非抢占，时间片大小为2---
第1个时间片被进程A使用,正在运行.....
    时间片使用完，所需时间4,使用时间2,还需时间2,进程A就绪
第2个时间片被进程B使用,正在运行.....
    时间片使用完，所需时间3,使用时间2,还需时间1,进程B就绪
第3个时间片被进程C使用,正在运行.....
    时间片使用完，所需时间5,使用时间2,还需时间3,进程C就绪
第4个时间片被进程A使用,正在运行.....
    时间片使用完，所需时间2,使用时间2,还需时间0,进程A结束
第5个时间片被进程D使用,正在运行.....
    时间片使用完，所需时间2,使用时间2,还需时间0,进程D结束
第6个时间片被进程E使用,正在运行.....
    时间片使用完，所需时间4,使用时间2,还需时间2,进程E就绪
第7个时间片被进程B使用,正在运行.....
    时间片使用完，所需时间1,使用时间2,还需时间-1,进程B结束
第8个时间片被进程C使用,正在运行.....
    时间片使用完，所需时间3,使用时间2,还需时间1,进程C就绪
第11个时间片被进程E使用,正在运行.....
    时间片使用完，所需时间2,使用时间2,还需时间0,进程E结束
第13个时间片被进程C使用,正在运行.....
    时间片使用完，所需时间1,使用时间2,还需时间-1,进程C结束
------所有进程调度完毕------------
```

- 笔算推导：





时刻	队列
0	A
2	BCA
4	CADEB
6	ADEBC
8	DEBC
10	EBC
12	BCE
13	CE
15	EC
17	C
18	C完

- **算法分析：**

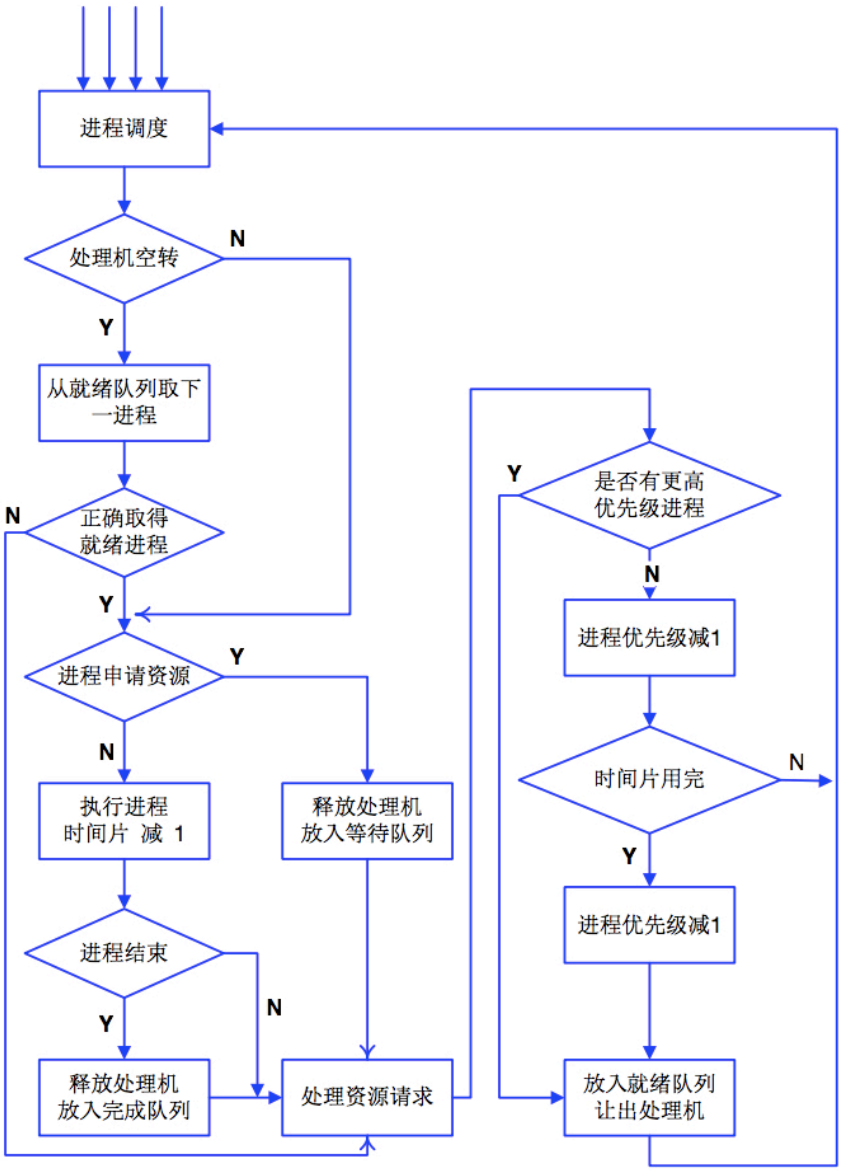
  在时间片轮转算法中，时间片大小对系统性能有很大的影响，如选择很小的时间片将有利于短作业，因为它能较快的完成，但会频繁的发生中断、进程上下文的切换，从而增加系统的开销；反之，如选择较长的时间片，使得每个进程都能在一个时间片内完成，时间片轮转算法便退化为 FCFS 算法，无法满足交互式用户的需求。

5. 多级反馈队列算法 MRLA：

- **数据：例题三数据 pcbdata3**

```
PCB pcbdata3[10]={
    //例程三数据
    {1000,"A",0,7,7,0,'R'},
    {1001,"B",5,4,4,0,'R'},
    {1002,"C",7,13,13,0,'R'},
    {1003,"D",12,9,9,0,'R'},
};
```

- **算法流程图**



核心代码：

各级时间片长度分别为 2，4，8

```
printf("---多级反馈算法调度：抢占式，时间片大小为2---\n");
        for(i=0;i<num3;i++)
        {
            t_ready[i]=ready[i];
        }
        time=order[0];
        for(l=0;done<num3;l++){
            //判断到达的进程数
            for(i=0;i<num3&&pcbdata3[ready[i]].time_start<=time;i++)
                temp_num=i+1;
```

```
11.
12.             if(time!=order[0]){
13.                 for(i=0;i<temp_num;i++) //按队列优先级排序
14.                     for(j=1;j<temp_num-i;j++){
15. if(pcbdata3[t_ready[j-1]].state=='F'||(queue[j-1]>queue[j]&&pcbdata3[t_re
    ady[j]].state!='F')){
16.                             temp=queue[j-1];
17.                             queue[j-1]=queue[j];
18.                             queue[j]=temp;
19.
20.                             temp=t_ready[j-1];
21.                             t_ready[j-1]=t_ready[j];
22.                             t_ready[j]=temp;
23.
24.                             temp=qtime[j-1];
25.                             qtime[j-1]=qtime[j];
26.                             qtime[j]=temp;
27.                         }
28.                     }
29.             }
30.             if(pcbdata3[t_ready[0]].state!='F'){
31.                 printf("队列%d 中的进程%s 占用
    CPU,",queue[0], pcbdata3[t_ready[0]].name);
32.                 printf("正在运行……\n    ");
33.                 _sleep(1);
34.                 if(!qtime[0])       //判断是否有未用完的时间片
35.                     qtime[0]=pow(2,queue[0]);
36.                 else
37.                     printf("继续使用时间片，");
38.                 for(i=1;i<qtime[0];i++)
39.                 {
40.                     time++;
41.
42.                     for(j=0;j<num3&&pcbdata3[ready[j]].time_start<=time;j+
    +)
43.                         temp_num2=j+1;
44.     //判断是否有进程进入比本进程更高优先级的队列
45.                     if(temp_num!=temp_num2&&queue[0]>queue[temp_num2-1]&&
    pcbdata3[t_ready[0]].time_left-i>0){
46.                         qtime[0]-=i;
47.                         break;
48.                     }
49.                 }
50.                 if(temp_num!=temp_num2&&queue[0]>queue[temp_num2-1]&&pcbda
```

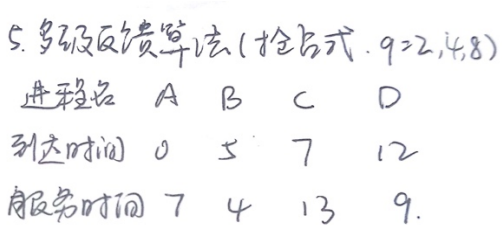
```
ta3[t_ready[0]].time_left-i>0){
51.     printf("发生抢占，使用时间片%d，剩余时间片%d，返回队列尾部\n",i,qtime[0]);
52.
53.                 }
54.                 else{
55.                     printf("时间片使用完，所需时间%d,", pcbdata3[t_ready[0]].time_left);
56.                     time++;
57.                     pcbdata3[t_ready[0]].time_used+=pow(2,queue[0]);
58.                     pcbdata3[t_ready[0]].time_left-=pow(2,queue[0]);

59.                     if(pcbdata3[t_ready[0]].time_left<=0){
60.                         printf("使用时间%d,还需时间%d,进程%s 结束\n",qtime[0], pcbdata3[t_ready[0]].time_left,pcbdata3[t_ready[0]].name);

61.                         done++;
62.                         pcbdata3[t_ready[0]].state='F';
63.                     }
64.                     else
65.                         printf("使用时间%d,还需时间%d,进程%s 进入队列%d 就绪\n",qtime[0],pcbdata3[t_ready[0]].time_left,pcbdata3[t_ready[0]].name,++queue[0]);
66.                     qtime[0]=0;
67.                 }
68.
69.             }
70.             for(j=1;j<temp_num2;j++){       //将执行的程序返回队尾排队
71.                 temp=queue[j];
72.                 queue[j]=queue[j-1];
73.                 queue[j-1]=temp;
74.
75.                 temp=qtime[j];
76.                 qtime[j]=qtime[j-1];
77.                 qtime[j-1]=temp;
78.
79.                 temp=t_ready[j];
80.                 t_ready[j]=t_ready[j-1];
81.                 t_ready[j-1]=temp;
82.             }
83.         }
84.         printf("------所有进程调度完毕-------------\n");
```

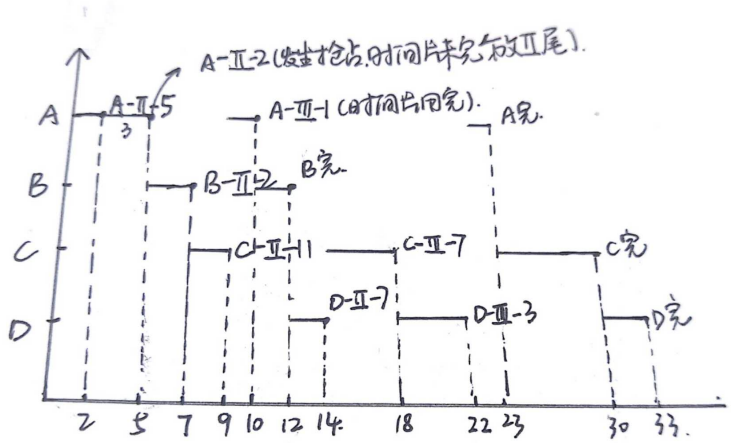
```
85. }
```

运行结果：

```
(4)时间片轮转Timeslice
(5)多级反馈队列MRLA
(0)退出
请输入上述一个数字：5
---多级反馈算法调度：抢占式，时间片大小为2---
队列1中的进程A占用CPU,正在运行……
    时间片使用完，所需时间7,使用时间2,还需时间5,进程A进入队列2就绪
队列2中的进程A占用CPU,正在运行……
    发生抢占，使用时间片3，剩余时间片1，返回队列尾部
队列1中的进程B占用CPU,正在运行……
    时间片使用完，所需时间4,使用时间2,还需时间2,进程B进入队列2就绪
队列1中的进程C占用CPU,正在运行……
    时间片使用完，所需时间13,使用时间2,还需时间11,进程C进入队列2就绪
队列2中的进程A占用CPU,正在运行……
    继续使用时间片，时间片使用完，所需时间5,使用时间1,还需时间1,进程A进入队列3就绪
队列2中的进程B占用CPU,正在运行……
    时间片使用完，所需时间2,使用时间4,还需时间-2,进程B结束
队列1中的进程D占用CPU,正在运行……
    时间片使用完，所需时间9,使用时间2,还需时间7,进程D进入队列2就绪
队列2中的进程C占用CPU,正在运行……
    时间片使用完，所需时间11,使用时间4,还需时间7,进程C进入队列3就绪
队列2中的进程D占用CPU,正在运行……
    时间片使用完，所需时间7,使用时间4,还需时间3,进程D进入队列3就绪
队列3中的进程A占用CPU,正在运行……
    时间片使用完，所需时间1,使用时间8,还需时间-7,进程A结束
队列3中的进程C占用CPU,正在运行……
    时间片使用完，所需时间7,使用时间8,还需时间-1,进程C结束
队列3中的进程D占用CPU,正在运行……
    时间片使用完，所需时间3,使用时间8,还需时间-5,进程D结束
------所有进程调度完毕------------
```

笔算推导：





时刻	队列I	队列II	队列III
0	A	/	/
2	/	A	/
5	B	A	/
7	C	AB	/
9	/	ABC	/
10	/	BC	A
12	D	C	A
14	/	CD	A
18	/	D	AC
22	/	/	ACD
23	/	/	CD
30	/	/	D
33	/	/	/

算法分析：

多级反馈队列调度算法的性能：
终端型作业用户一般能在第一队列完成，响应时间短，满足用户交互型需求；短批处理作业用户一般能在第二队列，至多第三队列可以完成，经历队列少，等待时间也少；长批处理作业用户可能经过多个队列才能完成，但在每个队列都能够得到时间片的分配，不会出现长时间不处理的情况。

6. 主函数

```
int  main()
{
    int i=0,sch=99;
    while(sch!=0)
    {printf("\n 请选择其中一种调度算法： \n");
    printf("(1)先来先服务 FCFS\n");
    printf("(2)短作业优先 SJF\n");
```

```
8.      printf("(3)高响应比 HRF\n");
9.      printf("(4)时间片轮转 Timeslice\n");
10.     printf("(5)多级反馈队列 MRLA\n");
11.     printf("(0)退出\n");
12.     printf("请输入上述一个数字: ");
13.     scanf("%d",&sch);
14.     switch(sch)
15.     {
16.     case 1:FCFS();break;
17.     case 2:SJF();break;
18.     case 3:HRF();break;
19.     case 4:Timeslice();break;
20.     case 5:MRLA();break;
21.     case 0:printf("退出程序\n");break;
22.     }
23.     }
24.     _keygo();
25.     return 0;
26. }
```

四、实验结论：（提供运行结果，对结果进行探讨、分析、评价，并提出结论性意见和改进想法）

五种调度算法及其实现思想

- **先来先服务算法：**

  按到达时间先后，选择最先来的作业最先执行

  对作业的到达时间按大小进行排序，然后按顺序执行

- **短作业优先算法：**

  在后备队列中，选择服务时间最短的作业最先执行

  对作业按到达时间排序，接着对到达的作业，即后备队列中的作业按服务时间排序，取服务时间最小的作业最先执行

- **高响应比算法：**

  对作业的优先权（响应时间/要求服务时间）进行计算，对优先权最高的最先执行

  计算后备队列中作业的优先权，并排序，优先权最高的最先执行

- **时间片轮转算法：**

  将所有就绪进程按先来先服务排成队列，把 CPU 分配给队首进程，进程只执行一个时间片，时间片用完后，将已使用时间片的进程送往就绪队列的末尾，分配处理机给就绪队列中下一进程

  将作业按到达时间排序，在后备队列中选择第一个作业，把 CPU 分配给它，执行一个时间片，时间片用完后，将作业送往后备队列的末尾，把 CPU 分配给下一个作业，直到所有作业完成

- **多级反馈队列调度算法：**

  设置多个就绪队列，各个队列优先级逐个降低，各个队列时间片逐个增加，优先级越高的队列执行时间片就越短，一般时间片按倍增规则，每个新进程首先进入第一个队列，遵循 FCFS，在当前队列的时间片内，进程若能完成，退出，进程若未完成，降级到第二个队列，同样遵循 FCFS 依次类推，若在第二个队列的时间片内仍未完成，再降级到第三个队列……

  设置多个就绪队列，各个队列优先级逐个降低，各个队列时间片逐个增加，优先级越高的队列执行时间片就越短，一般时间片按倍增规则， 例如，

第二队列的时间片要比第一个队列的时间片长一倍，……，第 i+1 个队列的时间片要比第 i 个队列的时间片长一倍，整合了时间片、 FCFS、优先级三种机制。

五、实验体会：（根据自己情况填写）

通过本次实验，基本了解计算机 CPU 对执行进程时的运作模式，掌握了先来先服务算法、短作业优先算法、高响应比算法、时间片轮转算法、多级反馈队列算法这五种 CPU 调度算法，掌握了这五种算法的的基本原理和运作机制，以及其算法实现。通过对这五种算法的模拟实验，对这五种调度算法有了更进一步的理解。

注：“指导教师批阅意见”栏请单独放置一页