实验二 作业调度

- 课程：操作系统
- 学院：信息工程学院
- 专业班级：计科2002
- 学号：2020010505
- 姓名：薛飞宇

一、实习目的

1. 模拟单处理器情况下的作业调度
2. 加深了解作业调度的工作原理
3. 理解作业控制块JCB的原理及数据结构
4. 理解"先来先服务"调度算法
5. 理解"短作业优先"调度算法
6. 阅读Linux源码，加深进程调度的理解

二、实习内容

任务1：代码阅读并调试实验

任务2：设计测试实验

任务3：生产者消费者实验

三、实习任务及完成情况

任务1：代码阅读并调试实验

1. **阅读下面源代码，完善程序空白处内容。**

    1. `p = p->next;`

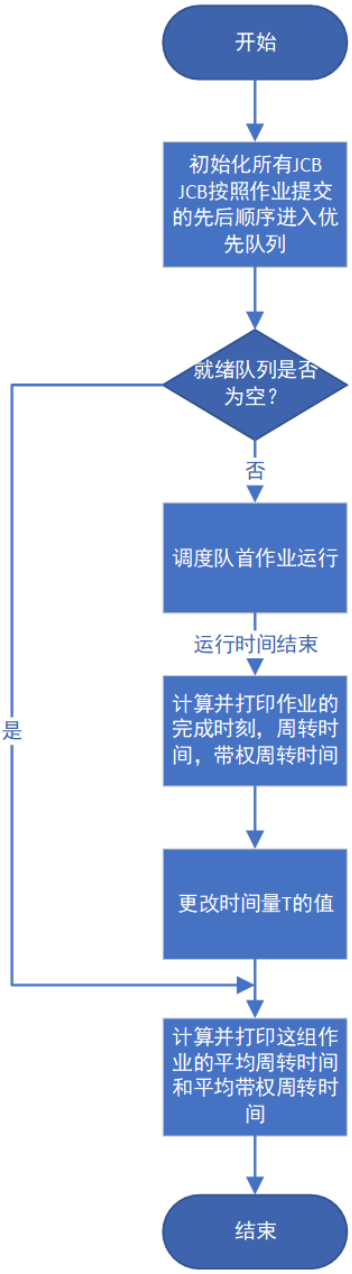
```
do{
    if(p->state=='W' && p->reachtime <= times)
        if(iden)
        {
            min=p;
            iden=0;
        }
        else if(p->needtime < min->needtime) min=p;
        //1根据上下文，填入一条语句
        p = p->next;
} while (p != NULL);
```

    2. `times++;`

```
if (iden)
{
    i--;
    printf("\n没有满足要求的进程,需等待");
```

```
    // 2根据上下文，填入一条语句
    times++;
    if (times>100)
    {
        printf("\n时间过长");
        getchar();
    }
}
else
{
    running(p, m);
}
```

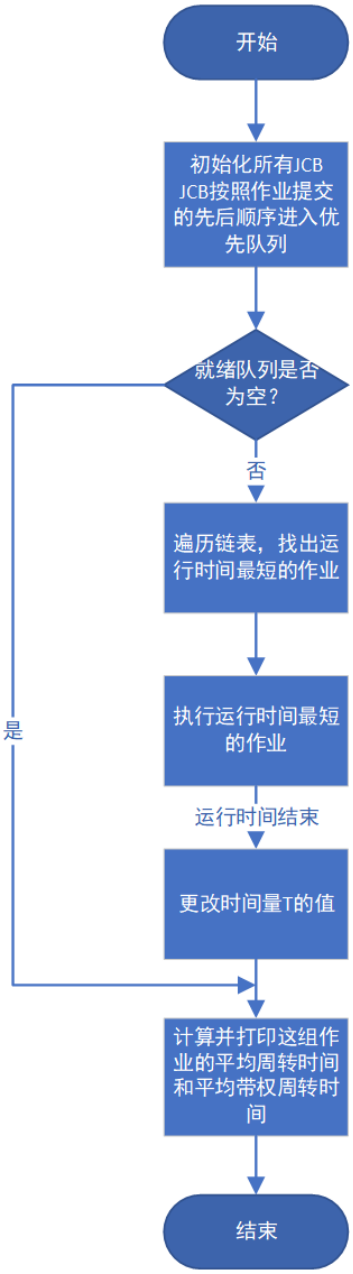
2. **阅读代码，写出源程序采用什么调度算法、算法流程图和程序功能。**
   - 调度算法1：先来先服务
     - 算法流程图：



     - 程序功能：

       模拟了先来先服务调度算法调度作业的过程
   - 调度算法2：短作业优先

- 算法流程图



- 程序功能：

  模拟了先来先服务调度算法和短作业优先调度算法的调度作业的过程

3. **解释作业控制块构JCB的定义和作用。**
   - JCB定义

```
struct jcb              //作业控制块
{
    char name[10];      //作业名
    int reachtime;      //提交时间
    int starttime;      //开始时间
    int needtime;       //运行需要的时间
    int finishtime;     //作业完成时间
    float cycletime;    //作业周转时间
    float cltime;       //带权周转时间
    char state;         //作业状态
    struct jcb *next;   //结构体指针
}
typedef struct jcb JCB;
```

- JCB作用：管理和调度作业，是作业在系统中存在的标志，保存了系统对作业进行管理和调度所需的全部信息。通常在JCB中包含的内容有：作业标识、用户名称、用户账号、作业类型、作业状态、调度信息、资源需求、资源使用情况等。在多道批处理系统中，为了管理和调度作业，操作系统通过JCB了解到作业要求，并分配资源和控制作业中程序和数据的编译、链接、装入和执行等。

4. **为main()写出每行的注释。**

```
int main()
{
    menu();              //调用菜单显示函数
    system("pause");     //menu()执行结束后清空控制台信息
    return 0;            //main函数返回0
}
```

5. **调试并运行代码，写出结果。**

    1. 代码

```
#define _CRT_SECURE_NO_WARNINGS 1
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<conio.h>
#define getpch(type) (type*)malloc(sizeof(type))
int n;
float T1=0,T2=0;
int times=0;

struct jcb               //作业控制块
{
    char name[10];       //作业名
    int reachtime;       //提交时间
    int starttime;       //开始时间
    int needtime;        //运行需要的时间
    int finishtime;      //作业完成时间
    float cycletime;     //作业周转时间
    float cltime;        //带权周转时间
    char state;          //作业状态
    struct jcb *next;    //结构体指针
}*ready=NULL,*p,*q;

typedef struct jcb JCB;

//建立作业控制块队列,先将其排成先来先服务的模式队列
void inital()
{
    int i;
    printf("\n输入作业数:");
    scanf("%d", &n);
    for (i = 0; i<n; i++)
    {
        p = getpch(JCB);
        printf("\n输入作业名:");
        scanf("%s", p->name);
        getchar();
        p->reachtime = i;
```

```
        printf("作业默认到达时间:%d", i);
        printf("\n输入作业要运行的时间:");
        scanf("%d", &p->needtime);
        p->state = 'W';
        p->next = NULL;
        if (ready == NULL)
            ready = q = p;
        else
        {
                q->next = p;
                q = p;
        }
    }
}

void disp(JCB*q,int m) //显示作业运行后的周转时间及带权周转时间等
{
    printf("\n作业%s正在运行，估计其运行情况：\n",q->name);
    printf("开始运行时刻：%d\n",q->starttime);
    printf("完成时刻：%d\n",q->finishtime);
    printf("周转时间：%f\n",q->cycletime);
    printf("带权周转时间：%f\n",q->cltime);
    getchar();
}

void running(JCB *p, int m)  //运行作业
{
    if (p == ready)      //先将要运行的作业从队列中分离出来
    {
        ready = p->next;
        p->next = NULL;
    }
    else
    {
        q = ready;
        while (q->next != p)  q = q->next;
        q->next = p->next;
    }
    p->starttime = times;     //计算作业运行后的完成时间,周转时间等等
    p->state = 'R';
    p->finishtime = p->starttime + p->needtime;
    p->cycletime = (float)(p->finishtime - p->reachtime);
    p->cltime = (float)(p->cycletime / p->needtime);
    T1 += p->cycletime;
    T2 += p->cltime;
    disp(p, m);          //调用disp()函数,显示作业运行情况
    times += p->needtime;
    p->state = 'F';
    printf("\n%s has been finished!\n", p->name);
    free(p);           //释放运行后的作业
    getchar();
}

void final() //最后打印作业的平均周转时间,平均带权周转时间
{
    float s,t;
    t=T1/n;
    s=T2/n;
```

```
    getchar();
    printf("\n\n作业已经全部完成!\n");
    printf("\n%d个作业的平均周转时间是：%f",n,t);
    printf("\n%d个作业的平均带权周转时间是%f：\n\n\n",n,s);
}

void sjf(int m)       // 最短作业优先算法
{
    JCB *min;
    int i,iden;
    system("cls");
    inital();                 //输入进程，先来先服务队列
    for(i=0;i<n;i++)
    {
        p=min=ready;
        iden=1;
        do{
                if(p->state=='W' && p->reachtime <= times)
                    if(iden)
                    {
                        min=p;
                        iden=0;
                    }
                    else if(p->needtime < min->needtime) min=p;
                //1根据上下文，填入一条语句
                p = p->next;
        } while (p != NULL);
        if(iden)
        {
            i--;
            times++;
            if (times>100)
            {
             printf("\nruntime is too long … error");
             getchar();
            }
        }
        else
        {
            running(min, m);
        }
    }
    final();
}

void fcfs(int m)      //先来先服务算法
{

    int i, iden;
    system("cls");
    inital();
    for (i = 0; i<n; i++)
    {

        p = ready;
        iden = 1;

        do {
```

```
            if (p->state == 'W' && p->reachtime <= times)
                iden = 0;
            if (iden)
                p = p->next;
        } while (p != NULL && iden);
        if (iden)
        {
            i--;
            printf("\n没有满足要求的进程,需等待");
            // 2根据上下文，填入一条语句
            times++;
            if (times>100)
            {
               printf("\n时间过长");
               getchar();
            }
        }
        else
        {
            running(p, m);
        }
    }
    final();
}

void menu()
{
    int m;
    system("cls");
    printf("\n\n\t\t**********************************************\t\t\n");
    printf("\t\t\t\t作业调度演示\n");
    printf("\t\t**********************************************\t\t\n");
    printf("\n\n\n\t\t\t1.先来先服务算法.");
    printf("\n\t\t\t2.最短作业优先算法.");
    printf("\n\t\t\t0.退出程序.");
    printf("\n\n\t\t\t\t选择所要操作:");
    scanf("%d", &m);
    switch (m)
    {
    case 1:
        fcfs(m);
        getchar();
        system("cls");
        break;
    case 2:
        sjf(m);
        getchar();
        system("cls");
        break;
        case 0:
        system("cls");
        break;
        default:
        printf("选择错误,重新选择.");
        getchar();
        system("cls");
        menu();
    }
```

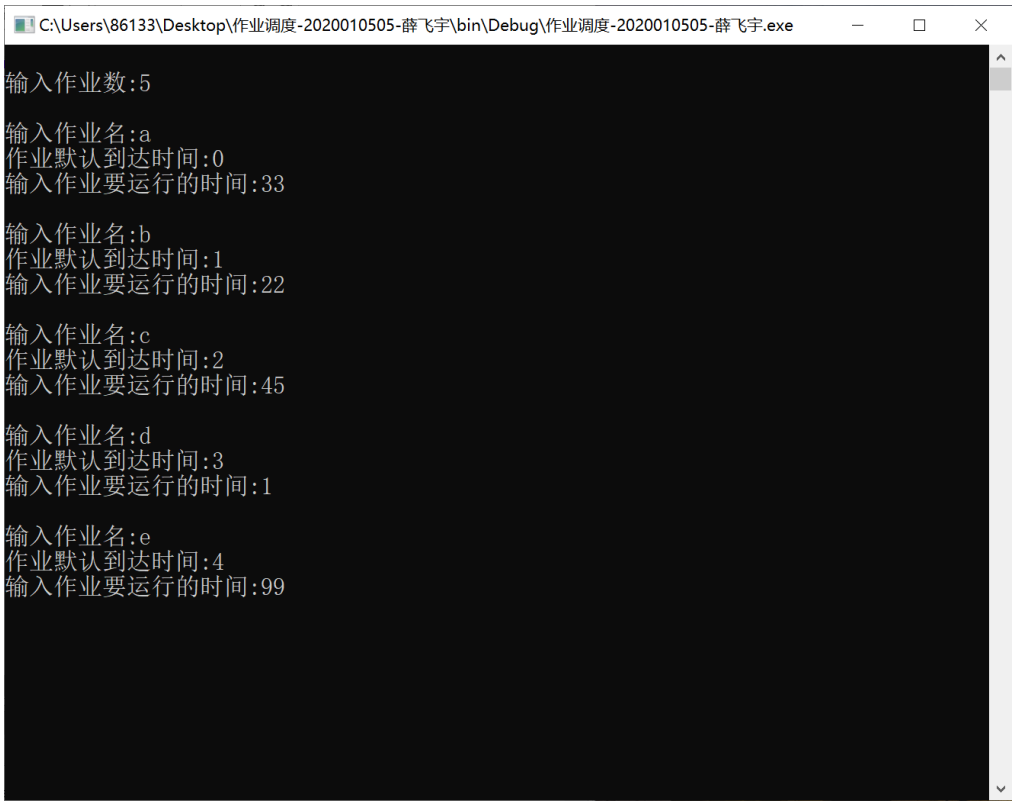
```
}
int main()
{
    menu();
    system("pause");
    return 0;
}
```

2. 调试结果

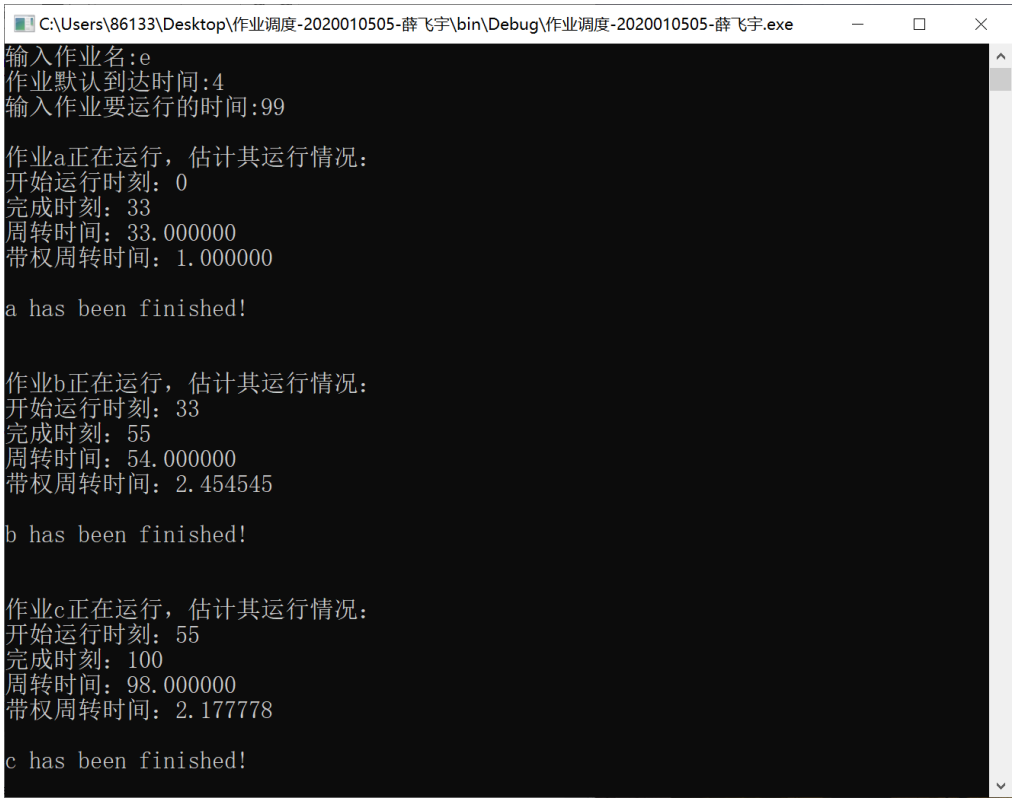
1. **先来先服务调度算法**

1.1 在菜单界面选择“先来先服务算法”

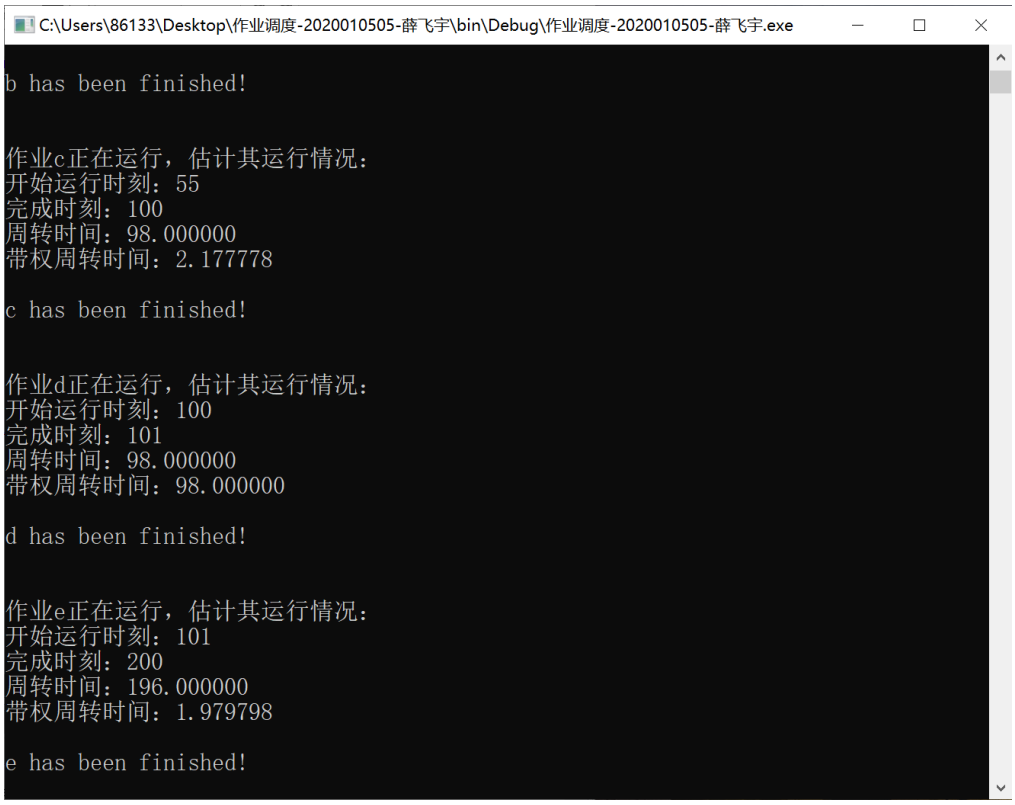
1.2 输入作业数和作业信息，如下图



1.3 时刻33，作业a运行结束；时刻55，作业b运行结束；时刻100，作业c运行结束

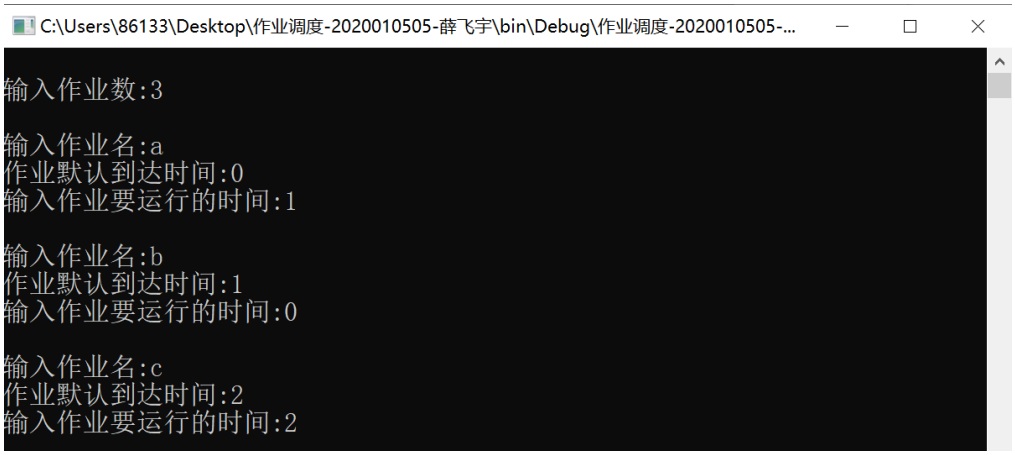


1.4 时刻101，作业d运行结束；时刻200，作业e运行结束

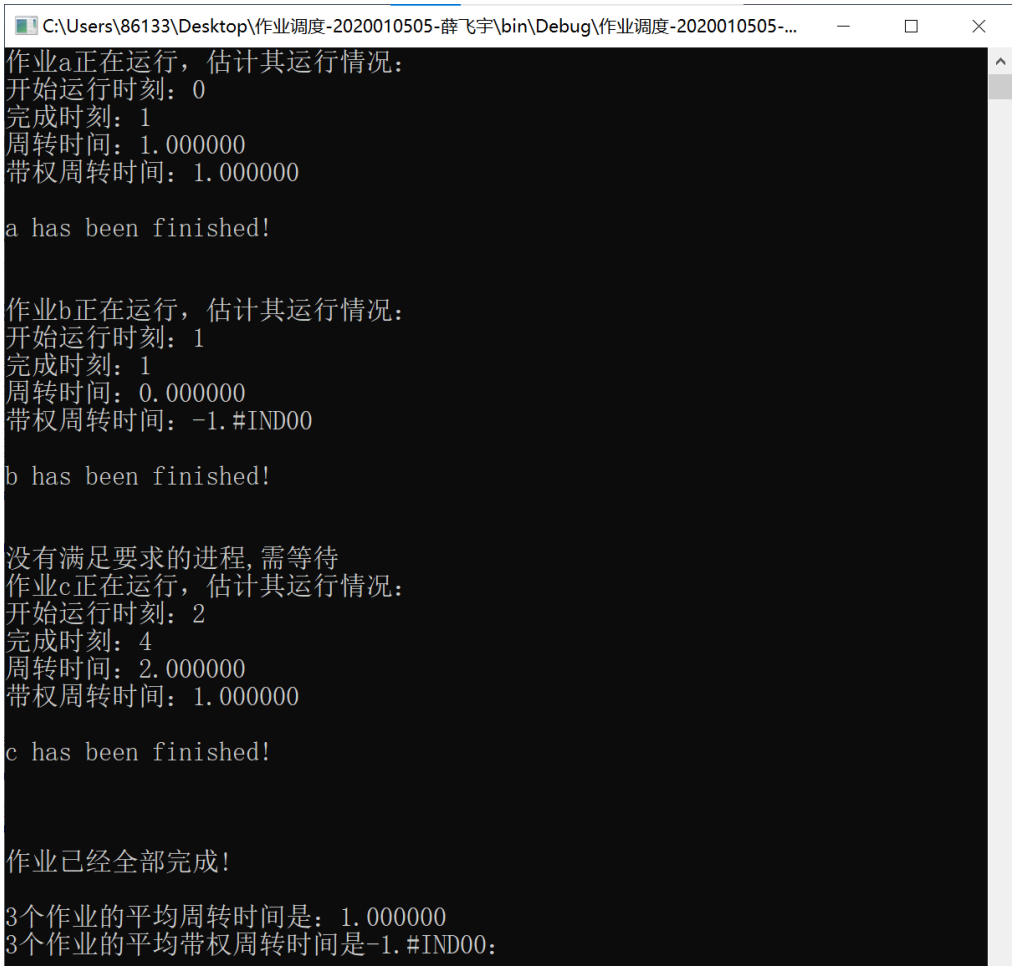


1.5 作业全部完成，平均周转时间为95，平均带权周转时间为21

1.6 重新安排3份做作业，其中作业b运行时间为0



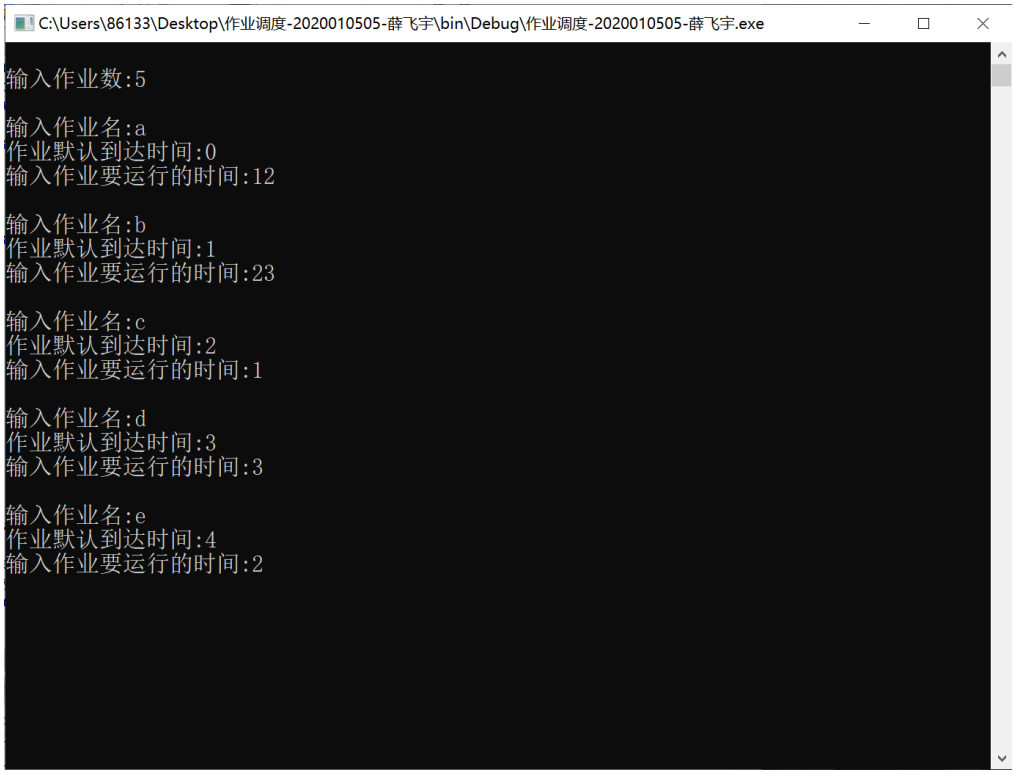
1.7 在时刻2没有满足要求的进程，处理机等待了1个时刻



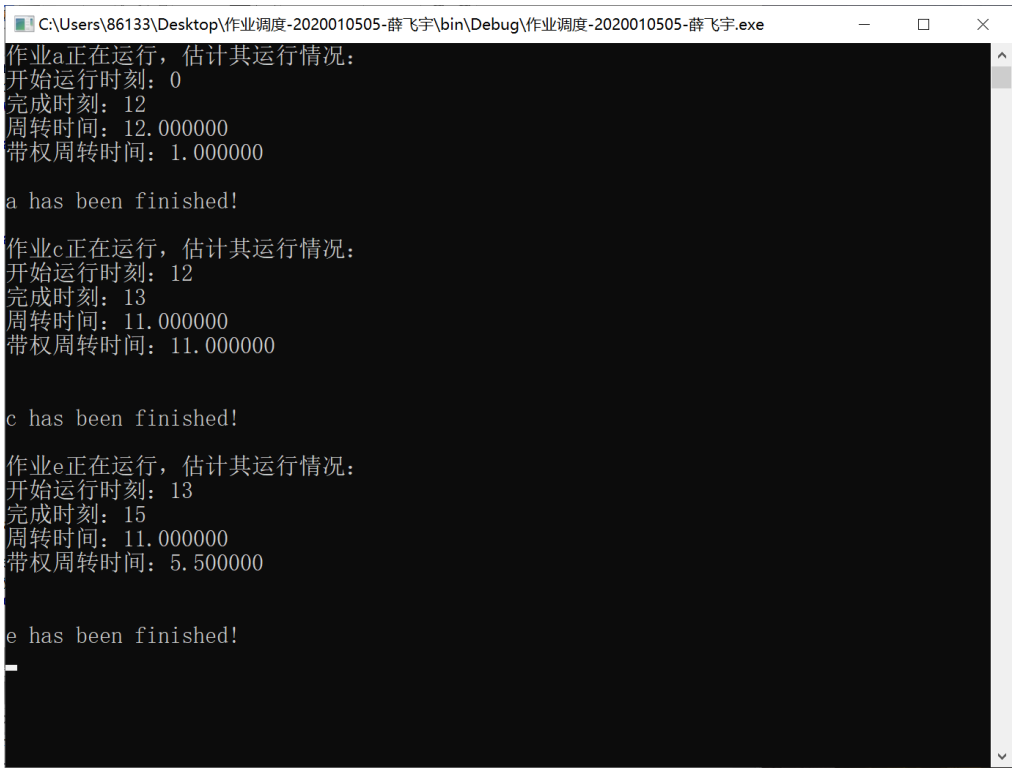
2. **“短作业优先算法”**

2.1 在菜单界面选择短作业优先算法

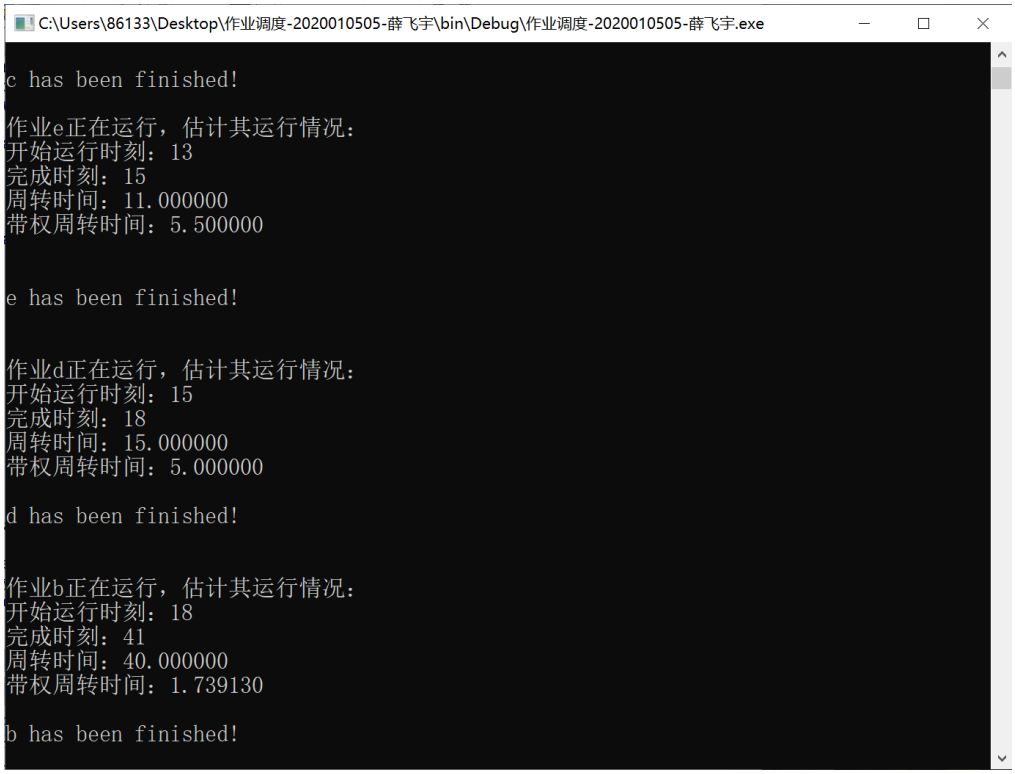
2.2 输入作业数和作业相关信息，如下图



2.3 时刻0，作业a接受调度；时刻12，由于c作业的运行时间最短，故最先接受调度；时刻13，作业e接受调度



2.4 时刻18，作业d运行完成；时刻41，作业b运行完成



2.5 时刻41，作业全部完成，平均周转时间为17.7，平均带权周转时间为4.8

任务2：设计测试实验

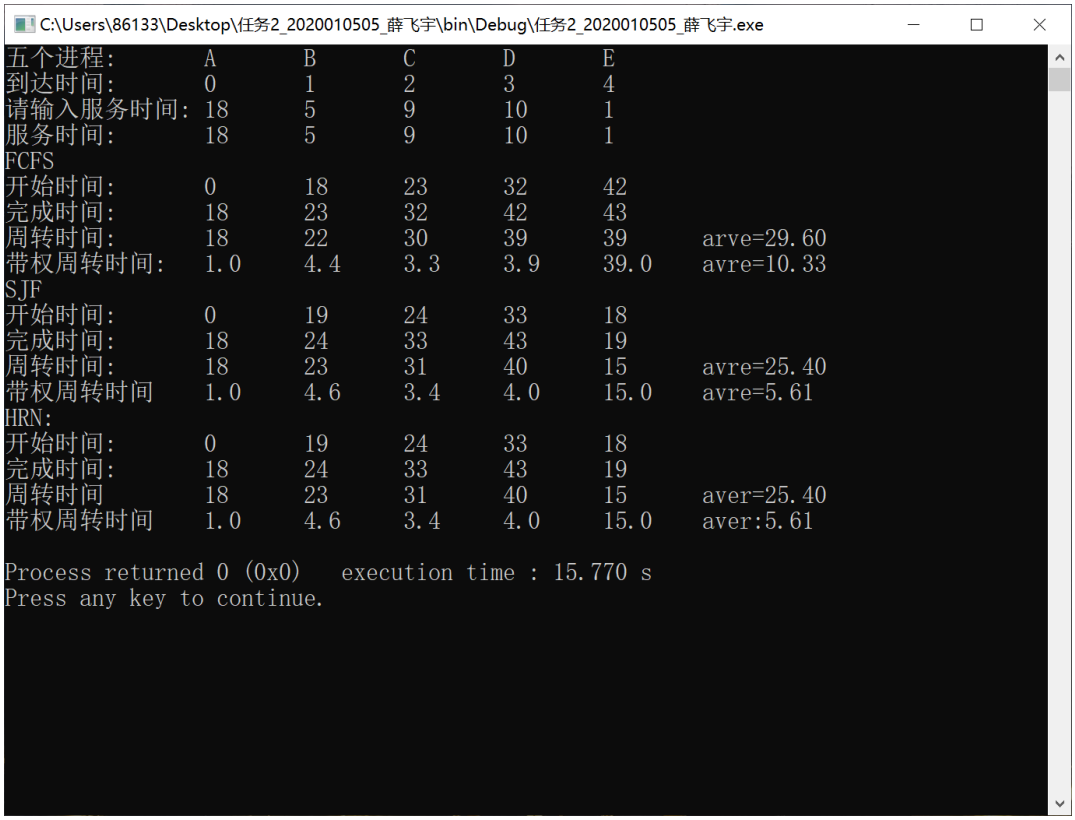
1. **编写并调试一个单道系统的作业调度模拟程序**

    1. 定义JCB,并操作之

```
struct jcb
{
    char name[10];       //作业名称
    int reachtime;       //作业抵达时间
    int starttime;       //作业开始时间
    int needtime;        //作业运行所需时间
    int finishtime;      //作业完成时间
```

```
    float cycletime;     //作业周转时间
    float cltime;        //带权周转时间
    char state;          //作业状态
    struct jcb *next;    //指向下一个作业的指针
    int Rp;              //响应比=响应时间/要求服务时间
    int waittime;        //等待时间
}
```

2. 描述作业队列
   - 先来先服务(FCFS)的作业队列按作业到达的先后顺序排列
   - 最短作业优先(SJF)的作业队列按作业由长到短的顺序排列
   - 响应比高者优先(HRN)的作业队列按照响应比由高到低的顺序排列
3. 对每种调度算法都要求打印每个作业开始运行时刻、完成时刻、周转时间、带权周转时间，以及这组作业的平均周转时间及带权平均周转时间,以比较各种算法的优缺点



1. 先来先服务（FCFS, First Come First Serve）是最简单的调度算法，按先后顺序进行调度。

   定义：

   按照作业提交或进程变为就绪状态的先后次序，分派CPU；

   当前作业或进程占用CPU，直到执行完或阻塞，才出让CPU（非抢占方式）。

   在作业或进程唤醒后（如I/O完成），并不立即恢复执行，通常等到当前作业或进程出让CPU。

   优点：

   比较有利于长作业

   有利于CPU繁忙的作业

   缺点：

   不利于短作业

   不利于I/O繁忙的作业

2. 短作业优先法（SJF, Shortest Job First）这是对FCFS算法的改进，其目标是减少平均周转时间。

   定义：

   对预计执行时间短的作业（进程）优先分派处理机。通常后来的短作业不抢先正在执行的作业。

   优点：

   比FCFS改善平均周转时间和平均带权周转时间，缩短作业的等待时间；

   提高系统的吞吐量；

   缺点：

   对长作业非常不利，可能长时间得不到执行；

   未能依据作业的紧迫程度来划分执行的优先级；

   难以准确估计作业（进程）的执行时间，从而影响调度性能。

3. 最高响应比优先法(HRN，Highest Response_ratio Next)是对FCFS方式和SJF方式的一种综合平衡。
   FCFS方式只考虑每个作业的等待时间而未考虑执行时间的长短，而SJF方式只考虑执行时间而未考虑等待时间的长短。因此，这两种调度算法在某些极端情况下会带来某些不便。HRN调度策略同时考虑每个作业的等待时间长短和估计需要的执行时间长短，从中选出响应比最高的作业投入执行。

   响应比R定义如下： R =(W+T)/T = 1+W/T

   其中T为该作业估计需要的执行时间，W为作业在后备状态队列中的等待时间。每当要进行作业调度时，系统计算每个作业的响应比，选择其中R最大者投入执行。这样，即使是长作业，随着它等待时间的增加，W / T也就随着增加，也就有机会获得调度执行。这种算法是介于FCFS和SJF之间的一种折中算法。

   优点：

   由于长作业也有机会投入运行，在同一时间内处理的作业数显然要少于SJF法，从而采用HRN方式时其吞吐量将小于采用SJF 法时的吞吐量。

   缺点：

   由于每次调度前要计算响应比，系统开销也要相应增加。

4. 源代码附件

```
#include<stdio.h>
#include<stdlib.h>
#define N 5

struct jcb
{
    char name[10];       //作业名称
    int reachtime;       //作业抵达时间
    int starttime;       //作业开始时间
    int needtime;        //作业运行所需时间
    int finishtime;      //作业完成时间
    float cycletime;     //作业周转时间
    float cltime;        //带权周转时间
    char state;          //作业状态
    struct jcb *next;    //指向下一个作业的指针
    int Rp;              //响应比=响应时间/要求服务时间
    int waittime;        //等待时间
```

```
}*ready = NULL, *p, *q;
int main()
{
        int a[2][5]={0};  //a[0][i]存放到达的时间，a[1][i]存放需要的服务时间
        int d[5]={0};
        int h[5]={0,1,2,3,4};
        double b[4][6]={0};  //FCFS
        double n[4][6]={0};  //SJF
        double m[4][6]={0};  //最高相应比优先
        double c[5];    //存放响应比
        double ha[5];   //存放服务时间用于响应比排序
        int i=0,j=0,t;
        int temp;
        int r,z;
        printf("五个进程:\tA\tB\tC\tD\tE\n");
        printf("到达时间:\t");
        for(i=0;i<N;i++)
        {
                a[0][i]=i;      //到达时间
                printf("%d\t",a[0][i]);
        }
        printf("\n");
        printf("请输入服务时间:\t");
        for(i=0;i<N;i++)
        {
            scanf("%d",&a[1][i]);  //服务时间
        }
        printf("服务时间:\t");
        for(i=0;i<N;i++)
        {
                d[i]=a[1][i];    //把服务时间顺便存放到d数组中
               printf("%d\t",a[1][i]);
        }
        printf("\n");
   /*先来先服务*/
   printf("FCFS\n");
   printf("开始时间:\t");
   b[0][0]=a[0][0];
   printf("%d\t",b[0][0]);
   for(i=1;i<N;i++)
   {
        b[0][i]=b[0][i-1]+(double)(a[1][i-1]);
        printf("%d\t",(int)b[0][i]);//b[0][i]存放的是各进程开始时间
   }
   printf("\n");
   b[1][0]=(double)a[1][0];
   printf("完成时间:\t%d\t",(int)b[1][0]); //b[1][i]存放的是完成时间
   for(i=1;i<N;i++)
   {
           b[1][i]=b[0][i]+(double)a[1][i];
           printf("%d\t",(int)b[1][i]);
   }
   printf("\n");
   printf("周转时间:\t");
   for(i=0;i<N;i++)
   {
           b[2][i]=b[1][i]-(double)a[0][i];
           b[2][5]+=b[2][i];   //b[2][5]周转时间累加和
```

```
            printf("%d\t",(int)b[2][i]); //b[2][i]存放的是周转时间
     }
     printf("arve=%.2f\n",b[2][5]/5);
     printf("带权周转时间:\t");
     for(i=0;i<N;i++)
     {
            b[3][i]=b[2][i]/(double)(a[1][i]);
            b[3][5]+=b[3][i];  //b[3][5]带权周转时间累加和
            printf("%.1f\t",(double)b[3][i]);//b[3][i]存放的是带权周转时间
     }
     printf("avre=%.2f\n",b[3][5]/5);

    /*短作业优先*/
    printf("SJF\n");
    for(i=1;i<N;i++)
         for(j=i+1;j<N;j++)
             if(d[i]>d[j])
             {
                    t=d[j];
                    d[j]=d[i];
                    d[i]=t;
                    temp=h[j];
                    h[j]=h[i];
                    h[i]=temp;
              }
     //h[i]即为进程服务的顺序    for(i=0;i<5;i++)printf("%d\t",h[i]);
          printf("开始时间:\t");
    for(i=1;i<N;i++)
    {
         n[0][h[i]]=a[1][h[i-1]]+n[0][h[i-1]];
    }
    for(i=0;i<N;i++)
         printf("%d\t",(int)n[0][i]);
    printf("\n");
    printf("完成时间:\t");
    n[1][0]=a[1][0];
    for(i=1;i<N;i++)
         n[1][h[i]]=n[0][h[i]]+a[1][h[i]];
    for(i=0;i<N;i++)
         printf("%d\t",(int)n[1][i]);
    printf("\n");
    printf("周转时间:\t");
    //n[2][0]=n[1][0]-a[0][0];//刚开始总是先执行第一个来的
    for(i=0;i<N;i++)
          n[2][i]=n[1][i]-a[0][i];
    for(i=0;i<N;i++)
    {
            n[2][5]+=n[2][i];  //总周转时间
            printf("%d\t",(int)n[2][i]);
     }
       printf("avre=%.2f\n",n[2][5]/5);
    printf("带权周转时间\t");
          for(i=0;i<N;i++)
             n[3][i]=n[2][i]/a[1][i];
   for(i=0;i<N;i++)
   {
             n[3][5]+=n[3][i];  //总带权周转时间
             printf("%.1f\t",n[3][i]);
```

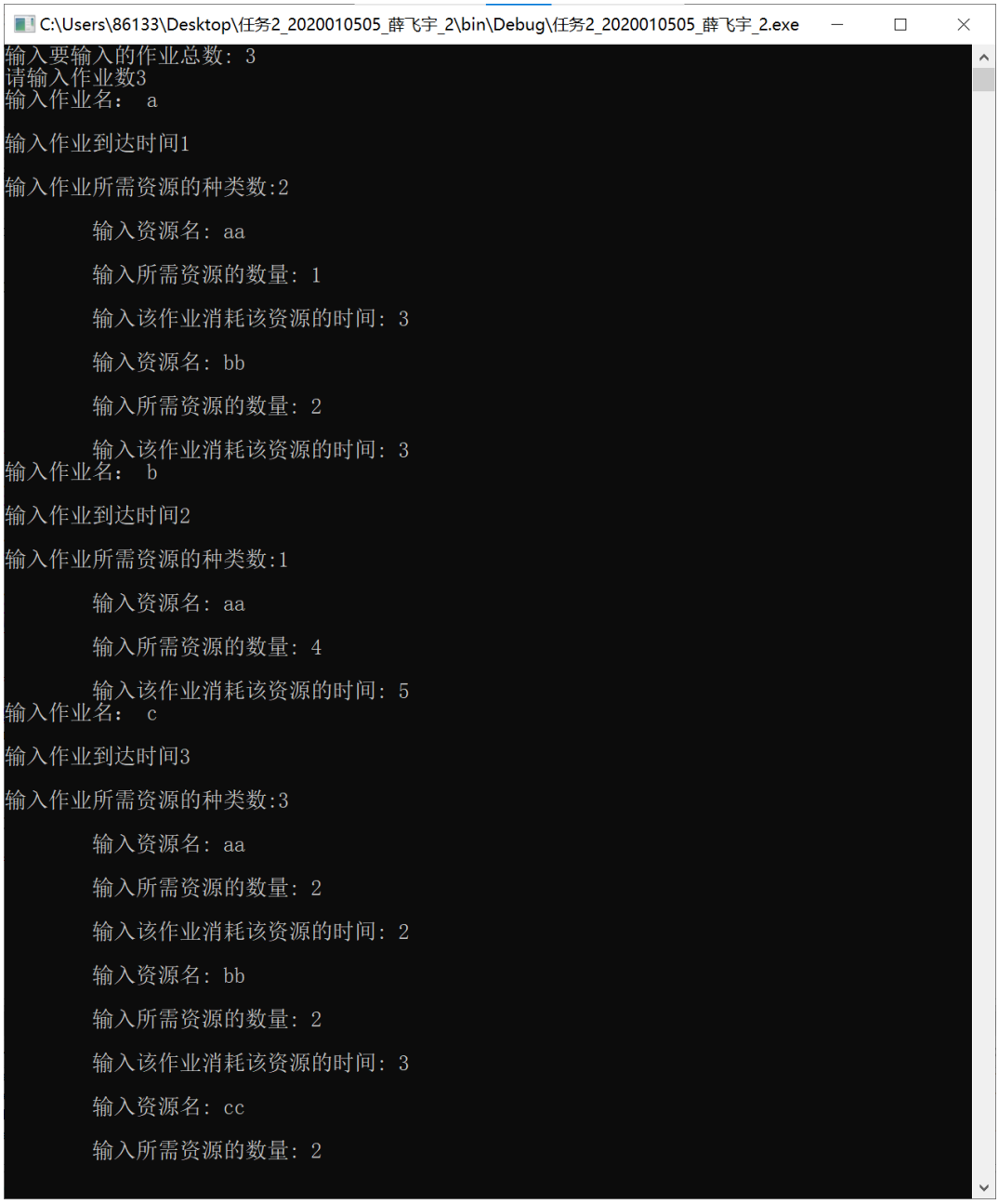
```
    }
      printf("avre=%.2f\n",n[3][5]/5);
    /*最高相应比优先*/
    printf("HRN:\n");
           for(i=1;i<N;i++)
        {
           ha[i]=a[1][i];//a[1][i]服务时间，ha[i]用于排序找响应比
         }
       m[0][0]=0;  //开始时间
       m[1][0]=a[1][0];
       r=0;
      for(int X=1;X<N;X++)
       {
         for(i=1;i<N;i++)
          {
               if(ha[i]!=0)
                  c[i]=1+(double)(m[1][r]-a[0][i])/a[1][i];//c[i]用于存放响
应比
             else
                c[i]=0;
          }
                  z=1;
                for(j=1;j<N;j++) //选出每次最高的响应比
                      {
                   if(c[z]<=c[j]&&ha[j]!=0)
                    z=j;
                      }
                   ha[z]=0;
                   m[0][z]=m[1][r];
                   m[1][z]=m[0][z]+a[1][z];
                   r=z;
         }
         printf("开始时间:\t");
         for(i=0;i<N;i++)
              printf("%d\t",(int)m[0][i]);
         printf("\n");
         printf("完成时间:\t");
         for(i=0;i<N;i++)
              printf("%d\t",(int)m[1][i]);
         printf("\n");
         printf("周转时间\t");
         for(i=0;i<N;i++)
         {
               m[2][i]=m[1][i]-a[0][i];
               m[2][5]+=m[2][i];  //m[2][5]周转时间累加和
               printf("%d\t",(int)m[2][i]);
         }
         printf("aver=%.2f\n",m[2][5]/5);
         printf("带权周转时间\t");
         for(i=0;i<N;i++)
         {
              m[3][i]=m[2][i]/a[1][i];
              m[3][5]+=m[3][i];
              printf("%.1f\t",m[3][i]);
          }
          printf("aver:%.2f\n",m[3][5]/5);
}
```

2. **编写并调试一个多道系统的作业调度模拟程序**

    1. 定义JCB,并操作之

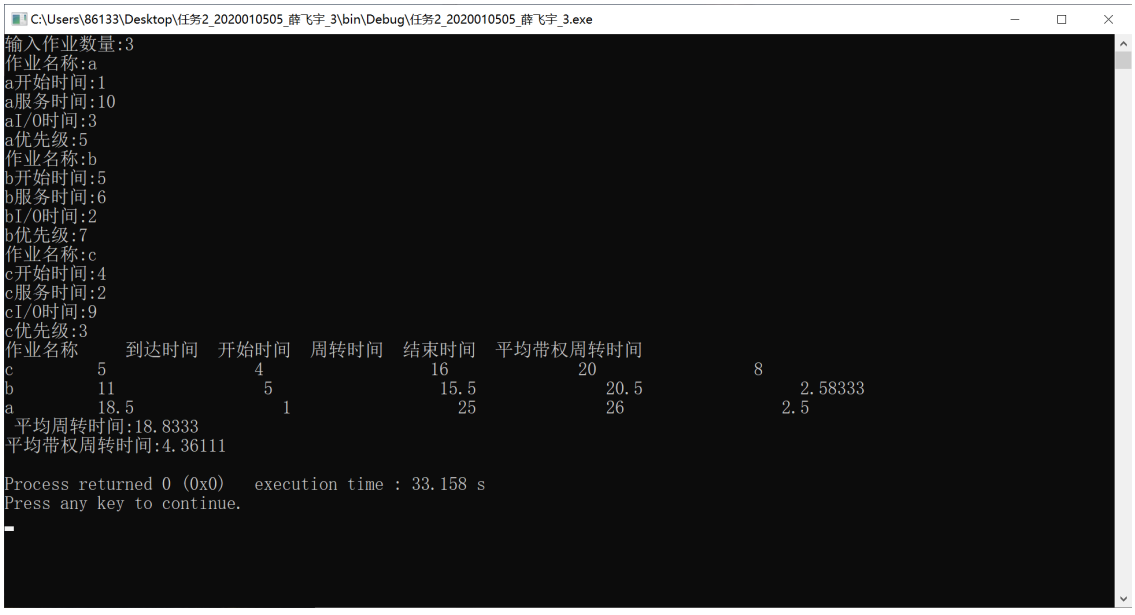
```
struct jcb      //作业控制块
{
    char name[10];
    int reachtime;
    int rnum;       //作业所需资源的种类数
    int starttime;
    int needtime;
    int finishtime;
    float cycletime;
    float cltime;   //带权周转时间
    char state;
    int IsUsed;    //标志是否使用该资源
    resource *res;  //所需资源指针
    struct jcb *next;       //结构体指针
}
```

    2. 描述作业队列
        - 先来先服务(FCFS)的作业队列按作业到达的先后顺序排列
    3. 定义多道系统中的各种资源种类及数量、调度作业时必须考虑到每个作业的资源要求
    4. 调试结果



3. **编写并调试一个多道系统的作业调度模拟程序**

调试结果



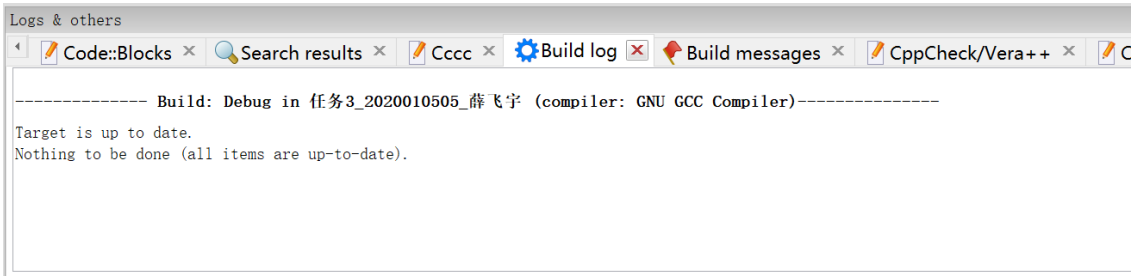
任务3：生产者消费者实验

1. 编辑源码，得到pthread.c

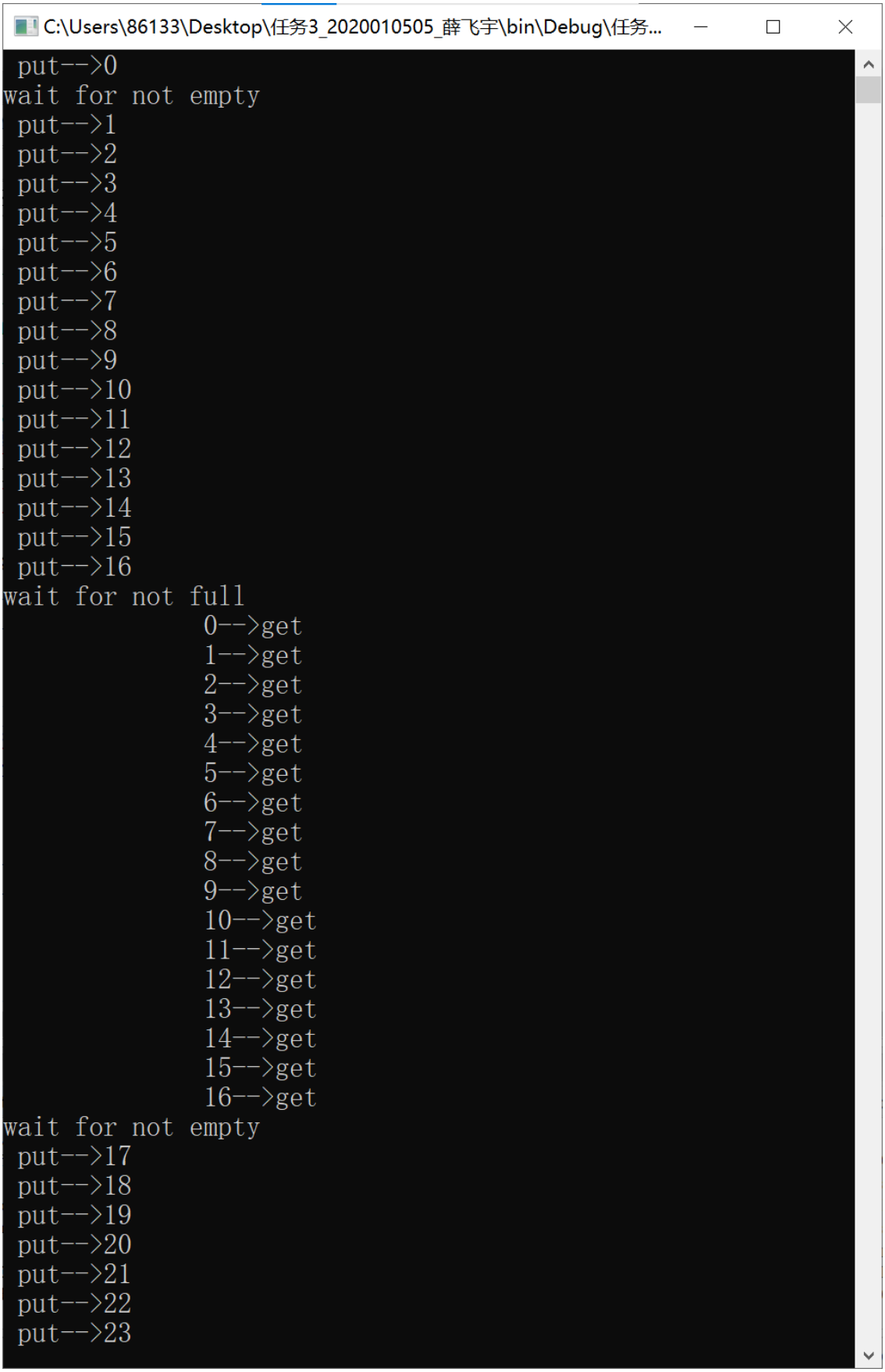
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "pthread.h"

#define BUFFER_SIZE 16
/*  设置一个整数的圆形缓冲区  */
struct prodcons {
  int buffer[BUFFER_SIZE];       /*  缓冲区数组  */
  pthread_mutex_t lock;          /*  互斥锁  */
  int readpos, writepos;        /*  读写的位置*/
  pthread_cond_t notempty;       /*  缓冲区非空信号  */
  pthread_cond_t notfull;        /*缓冲区非满信号  */
};

/*-------------------------------------------------------*/
/*初始化缓冲区*/
void init(struct prodcons * b)
{
  pthread_mutex_init(&b->lock, NULL);
  pthread_cond_init(&b->notempty, NULL);
  pthread_cond_init(&b->notfull, NULL);
  b->readpos = 0;
  b->writepos = 0;
}
/*-------------------------------------------------------*/
/*  向缓冲区中写入一个整数*/
void put(struct prodcons * b, int data)
{
        pthread_mutex_lock(&b->lock);       /*获取互斥锁*/

    /*等待缓冲区非满*/
    while ((b->writepos + 1) % BUFFER_SIZE == b->readpos) {
        printf("wait for not full\n");
        pthread_cond_wait(&b->notfull, &b->lock);  /*等待状态变量 b->notfull,
不满则跳出阻塞*/
    }
    /*写数据并且指针前移*/
    b->buffer[b->writepos] = data;
    b->writepos++;
    if (b->writepos >= BUFFER_SIZE) b->writepos = 0;
 /*设置缓冲区非空信号*/
    pthread_cond_signal(&b->notempty);     /*设置状态变量*/

    pthread_mutex_unlock(&b->lock);             /*//释放互斥锁*/
}
/*-------------------------------------------------------*/
/*从缓冲区中读出一个整数  */
int get(struct prodcons * b)
{
    int data;
    pthread_mutex_lock(&b->lock);                    /*获取互斥锁*/
    /*  等待缓冲区非空*/
```

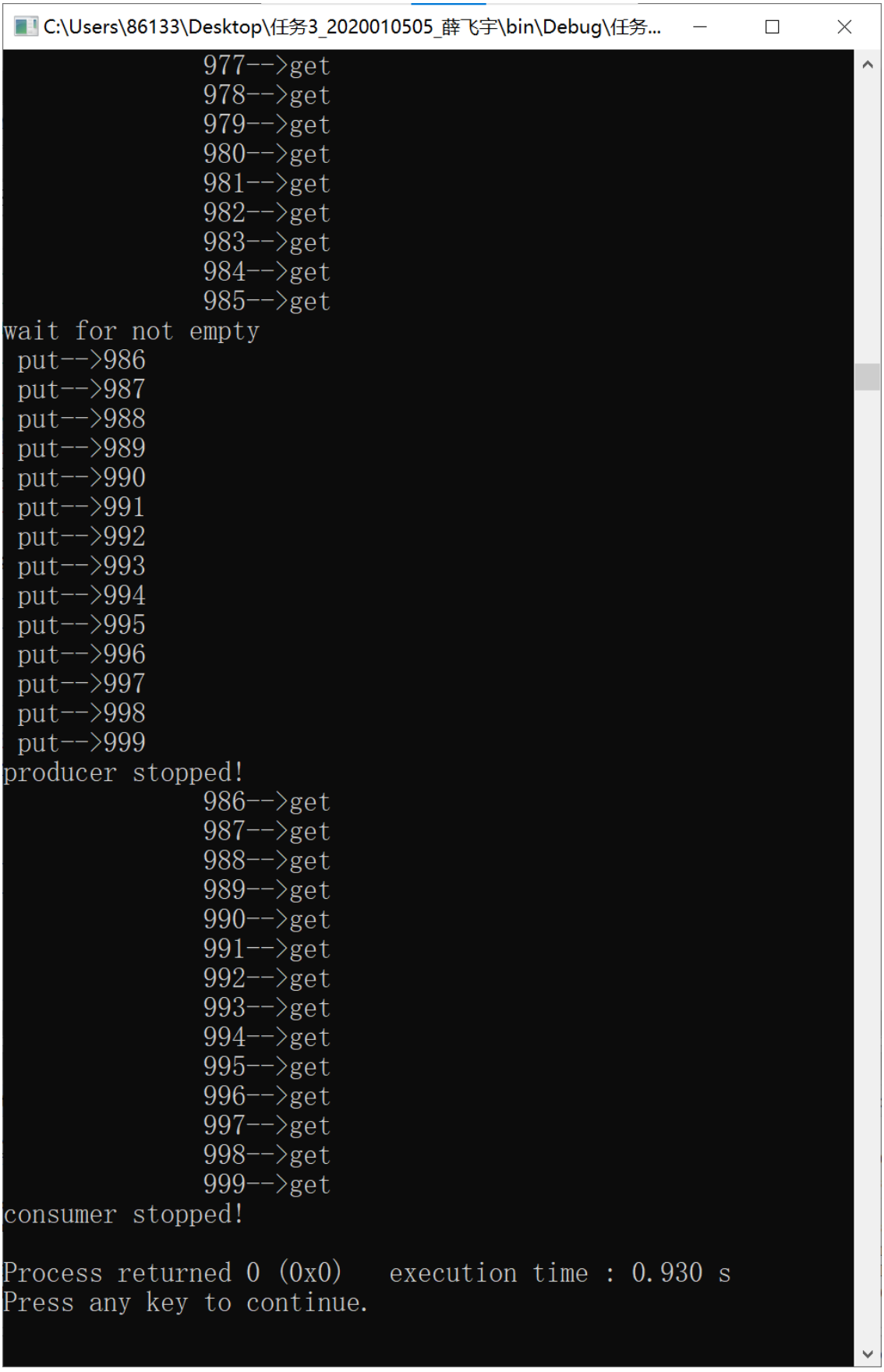
```
    while (b->writepos == b->readpos) {
        printf("wait for not empty\n");
        pthread_cond_wait(&b->notempty, &b->lock);
/*等待状态变量 b->notempty，不空则跳出阻塞。否则无数据可读 */
    }
    /*  读数据并且指针前移  */
    data = b->buffer[b->readpos];
    b->readpos++;
    if (b->readpos >= BUFFER_SIZE) b->readpos = 0;
    /*  设置缓冲区非满信号*/
    pthread_cond_signal(&b->notfull);     /*设置状态变量*/
    pthread_mutex_unlock(&b->lock);      /*释放互斥锁*/
    return data;
}
/*-------------------------------------------------------*/
#define OVER (-1)
struct prodcons buffer;
/*-------------------------------------------------------*/
void * producer(void * data)
{
    int n;
    for (n = 0; n < 1000; n++) {
        printf(" put-->%d\n", n);
        put(&buffer, n);
    }
  put(&buffer, OVER);
  printf("producer stopped!\n");
  return NULL;
}
/*-------------------------------------------------------*/
void * consumer(void * data)
{
  int d;
  while (1) {
    d = get(&buffer);
    if (d == OVER ) break;
    printf("              %d-->get\n", d);
  }
  printf("consumer stopped!\n");
  return NULL;
}
/*-------------------------------------------------------*/
int main(void)
{
    pthread_t th_a, th_b;
    void * retval;

    init(&buffer);
    pthread_create(&th_a, NULL, producer, 0);
    pthread_create(&th_b, NULL, consumer, 0);
/*  等待生产者和消费者结束  */
    pthread_join(th_a, &retval);
    pthread_join(th_b, &retval);

    return 0;
}
```

2. 使用gcc编译（或者使用make工具）



3. 运行


C:\Users\86133\Desktop\任务3_2020010505_薛飞宇\bin\Debug\任务...
put-->0
wait for not empty
put-->1
put-->2
put-->3
put-->4
put-->5
put-->6
put-->7
put-->8
put-->9
put-->10
put-->11
put-->12
put-->13
put-->14
put-->15
put-->16
wait for not full
0-->get
1-->get
2-->get
3-->get
4-->get
5-->get
6-->get
7-->get
8-->get
9-->get
10-->get
11-->get
12-->get
13-->get
14-->get
15-->get
16-->get
wait for not empty
put-->17
put-->18
put-->19
put-->20
put-->21
put-->22
put-->23



最终，生产者生产了1000个产品，全部被消费者取出。

四、实习总结

本次实验回顾了几个经典的作业调度算法，模拟单处理器情况下的作业调度，加深了解作业调度的工作原理，理解作业控制块JCB的原理及数据结构，自己动手写出了“先来先服务”调度算法。先来先服务（FCFS, First Come First Serve）是最简单的调度算法，按先后顺序进行调度，比较有利于长作业，有利于CPU繁忙的作业，不利于短作业，不利于I/O繁忙的作业。“短作业优先”调度算法，短作业优先法

（SJF, Shortest Job First）这是对FCFS算法的改进，其目标是减少平均周转时间，比FCFS改善平均周转时间和平均带权周转时间，缩短作业的等待时间，提高系统的吞吐量；对长作业非常不利，可能长时间得不到执行。最高响应比优先法(HRN，Highest Response_ratio Next)是对FCFS方式和SJF方式的一种综合平衡。由于长作业也有机会投入运行，在同一时间内处理的作业数显然要少于SJF法，从而采用HRN方式时其吞吐量将小于采用SJF 法时的吞吐量。最后自己尝试阅读Linux源码，执行"生产者消费者"问题的代码，加深进程调度的理解。本次实验圆满完成。