gdut

学号： 姓名： 协作者： 无

实验 二 题目\_作业调度 \_第 周星期\_\_

1. **实验目的**

作业管理是用户与操作系统的接口。作业调度的主要功能是检查系统是否能满足用户作业的资源要求以及按照一定的算法选取作业。

本实验的目的是通过模拟作业调度算法的设计加深对作业管理基本原理的理解。

1. **实验内容和要求**

* **实验内容**
* 为单道批处理系统设计一个作业调度程序

1. 编写并调试一个单道处理系统的作业调度模拟程序。
2. 作业调度算法：分别采用先来先服务（FCFS），最短作业优先（SJF）、响应比高者优先（HRN）的调度算法。
3. 由于在单道批处理系统中，作业一旦投入运行，它就占有计算机的一切资源直到作业完成为止，因此调度作业时不必考虑它所需要的资源是否得到满足，它所占用的 CPU时限等因素。
4. 每个作业由一个作业控制块JCB表示，JCB可以包含如下信息：作业名、提交时间、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)三种状态之一。每个作业的最初状态总是等待W。
5. 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，并比较各种算法的优缺点。

* 模拟批处理多道操作系统的作业调度

1. 写并调试一个作业调度模拟程序。
2. 作业调度算法：分别采用先来先服务（FCFS）和短作业优先调度算法。
3. 在批处理系统中，要假定系统中具有的各种资源及数量、调度作业时必须考虑到每个作业的资源要求，所需要的资源是否得到满足。

作业调度程序负责从输入并且选择若干个作业进入主存，为它们分配必要的资源，当它们能够被进程调度选中时，就可占用处理器运行。作业调度选择一个作业的必要条件是系统中现有的尚未分配的资源可满足该作业的资源要求。但有时系统中现有的尚未分配的资源既可满足某个作业的要求也可满足其它一些作业的要求，那么，作业调度必须按一定的算法在这些作业中作出选择。当作业正常运行完毕或因发生错误非正常终止时，作业进入完成状态，此时，系统将收回该作业所占用的全部资源，并清除有关的JCB。并输出显示作业运行情况及作业输出结果。

* **实验要求**

按照实验内容完成单道批处理调度模拟程序和模拟批处理多道操作系统的作业调度程序，画出程序流程图，调试自己完成的程序并且完成心得等实验内容。

**注：本次实验我所实现的是批处理多道操作系统的作业调度程序。**

1. **实验主要仪器设备和材料**

* **实验环境**

硬件环境:IBM-PC或兼容机

软件环境:VC++6.0编程环境

1. **实验原理及设计方案**

* **实验原理及设计思路**

本实验旨在模拟操作系统的作业调度情况，一般而言，有如下几种调度算法：先来先服务、最短作业优先、响应比高者优先。在单道环境下其调度是比较简单的，由于本实验我采用了多道批处理环境，所以下边的内容针对多道批处理环境。

本实验中，我分别用了两个结构体来表示作业控制块和进程控制块，作业的调度我采用的是先来先服务算法，但请注意，由于涉及到资源的分配，如果先来的作业得不到充足的资源，那么它是不会被调入内存的。而如果后来的作业所需的资源不大于系统当前所剩资源，则先调度后来的作业。其具体实现如下：创建一个作业控制块链表，每当新创建作业后，就将其作业控制块添加到该链表的末端，然后启动作业调度模块，从头到尾一个一个地扫描作业控制块，如果发现它所需求的资源（内存和磁带机数）系统可以予以满足，那么则为它创建一个进程控制块并插入到进程控制块多级队列当中，然后置该作业控制块的状态(state)为运行状态(R)。如此直到整个作业控制块扫描完毕。

本实验的进程调度采用了多级队列调度算法，由于实验一已经实现了，所以在此不再详细说明该方法，具体请看实验一的“实验原理及设计方案”。

另外，本系统还可以动态地添加作业，而每当添加完一个作业后，便自动调用作业调度模块，为其调度。

对于已完成的作业，我采用的做法是把它从作业控制块链中删除并添加到一个已完成作业链当中，这样可以很方便查看已经完成了哪些作业。而且在作业调度时不用再扫描这些已完成的作业，节省了一些时间。

* **数据结构的说明**

//作业控制块

struct Job

{

char userName[10]; //用户名

char jobName[10]; //作业名

char state; //作业状态,W表示后备，R表示执行，F表示完成

int reachTime[2]; //索引0表示小时,1表示分钟

int runTime; //所需的运行时间，以分钟为单位

int resourceMainMemory; //所需的内存空间，单位为KB

int resourceTapeNum; //所需的磁带机数

int order; //被选中运行的次数，数字越小表示越先被选中

JCB\* next;

}\*headJCB=NULL,\*endJCB=NULL,\*headJCBFish=NULL,\*endJCBFish;

//进程控制块

struct pcb{ //定义进程控制块PCB

char name[10];

char state;

int grade; //队列的级数, 最大值为MAXQUEUENUM

int ntime; //假定时间片以分钟为单位

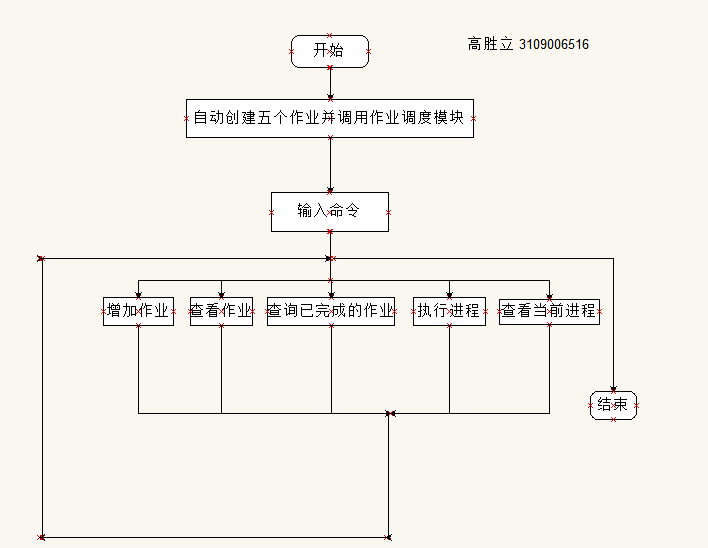
int rtime; //已经运行的时间

struct pcb\* link;

}\*head[MAXQUEUENUM],\*end[MAXQUEUENUM],\*p;

typedef struct pcb PCB;

* **程序流程图**



**增加作业模块：** **执行模块**



* **给出程序中源程序名和可执行程序名**
* **源程序文件**： SystemTwo.cpp
* **可执行文件**： SystemTwo.exe
* **程序清单**

#include <stdio.h>

#include <malloc.h>

#include <string.h>

#include <math.h>

#include<windows.h>

#define MAXQUEUENUM 10

#define getpch(type)(type\*)malloc(sizeof(type))

typedef struct Job JCB;

int tatalResourceMainMemory = 100;

int totalResourceTapeNum = 5;

int choiceOrder = 0;//作业被选中运行的次数

//作业控制块

struct Job

{

char userName[10];

//用户名

char jobName[10];

//作业名

char state;

//作业状态,W表示后备，R表示执行，F表示完成

int reachTime[2];

//索引0表示小时,1表示分钟

int runTime;

//所需的运行时间，以分钟为单位

int resourceMainMemory;

//所需的内存空间，单位为KB

int resourceTapeNum;

//所需的磁带机数

int order;

//被选中运行的次数，数字越小表示越先被选中

JCB\* next;

// 指针，指向下一个JCB

}\*headJCB=NULL,\*endJCB=NULL,\*headJCBFish=NULL,\*endJCBFish;

//进程控制块

struct pcb{

/\*定义进程控制块PCB\*/

char name[10];

//存储用户名

char state;

//状态

int grade;

//队列的级数, 最大值为MAXQUEUENUM

int ntime;

//假定时间片以分钟为单位

int rtime;

//已经运行的时间

struct pcb\* link;

}\*head[MAXQUEUENUM],\*end[MAXQUEUENUM],\*p;

typedef struct pcb PCB;

//进程控制块相关操作

void createPCB(JCB\* jcbObj)

//建立进程控制块函数,把ptr所指向进程插入第一队列的末尾

{

p=getpch(PCB);

strcpy(p->name,jcbObj->jobName);

p->state = 'w';

//初始化为后备状态

p->grade = 1;

//初始化队列级别为1

p->ntime = jcbObj->runTime;

p->rtime = 0;

//已运行的时间初始化为0

p->link = NULL;

//初始化没有连接下一个

}

//把新创建的PCB块添加到对应序号的PCB链当中

void addPCBIntoList(int index)

{

if(head[index]==NULL)

head[index] = end[index] = p;

else

{

end[index]->link = p;

end[index] = end[index]->link;

}

p = NULL;

}

void disp(PCB \*pr)

/\*显示特定PCB块的信息\*/

{

printf("\n 作业名\t 状态 \t 等级\t 需要时间\t 运行时间 \n");

printf("%s ",pr->name);

printf("%c ",pr->state);

printf("%d ",pr->grade);

printf("%d ",pr->ntime);

printf("%d ",pr->rtime);

printf("\n");

}

void check(PCB\* ptr)

/\*建立进程查看函数\*/

{

int j = 0;

printf("\n \*\*\*\*当前正在运行的进程是:%s",ptr->name);

/\*显示当前运行的进程\*/

//printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*当前进程信息\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

disp(ptr);

printf("\n\*\*\*\*当前就绪队列状态为:\n");

/\*显示就绪队列状态\*/

for(j=ptr->grade-1; j<MAXQUEUENUM; j++)

{

PCB\* temp = NULL;

if(j==ptr->grade-1)

//还为当前队列，要特殊考虑

{

if(head[j]->link == NULL)

continue;

temp = head[j]->link;

}

else if(head[j] == NULL)

//空队列，跳过

continue;

else

temp = head[j];

while(temp)

//打印出本PCB链的信息

{ disp(temp);

temp = temp->link;

}

}

}

//显示所有PCB队列中的PCB块的信息

void printAllProcesses()

{

int i;

for(i=0; i<MAXQUEUENUM; i++)

{

PCB\* temp = head[i];

while(temp)

{

disp(temp);

temp = temp->link;

}

}

}

//不同级数的进程时间片长度的计算

int runTime(int grade)

{

return (int)pow(2,grade-1);

}

//销毁进程及对应的作业

void jobInvoke();

//提前声明

void addFinishJob(JCB\* finishJCB);

//提前声明

void destroy()

{

char jobName[10];

JCB\* temp = headJCB;

JCB\* pre = NULL;

//一个辅助指针

printf("进程[%s]已经完成了,对应作业[%s]完成，已经从作业队列中删除......\n",p->name,p->name);

strcpy(jobName,p->name);

free(p);

p = NULL;

//从作业队列当中删除该进程对应的作业

while(temp)

{

if(strcmp(temp->jobName,jobName)==0)

//说明该作业已经完成，可删除

{

temp->state = 'F';

if(headJCB == endJCB)

{//说明只有一个作业

tatalResourceMainMemory += temp->resourceMainMemory;

totalResourceTapeNum += temp->resourceTapeNum;

temp->next = NULL;

addFinishJob(temp);

endJCB = NULL; printf("\naaa1\n");

}

else if(headJCB == temp)

{//说明第一个作业已经完成了

headJCB = headJCB->next;

tatalResourceMainMemory += temp->resourceMainMemory;

totalResourceTapeNum += temp->resourceTapeNum;

temp->next = NULL;

addFinishJob(temp); printf("\naaa2\n");

}else if(endJCB == temp)

{//说明最后一个作业完成了

endJCB = pre;

tatalResourceMainMemory += temp->resourceMainMemory;

totalResourceTapeNum += temp->resourceTapeNum;

temp->next = NULL;

addFinishJob(temp); printf("\naaa3\n");

}else

{//最一般的情况

pre->next = temp->next;

tatalResourceMainMemory += temp->resourceMainMemory;

totalResourceTapeNum += temp->resourceTapeNum;

temp->next = NULL;

addFinishJob(temp); printf("\naaa4\n");

}

}

pre = temp;

if(temp)

temp = temp->next;

}

jobInvoke();

}

//进程运行函数

void run()

{

int i;

for(i=0; i<MAXQUEUENUM; i++)

{

if(head[i] == NULL)

continue;

p = head[i];

check(p);

head[i] = head[i]->link;

if(head[i] == NULL)//若只有一个pcb块，则取走后尾指针应赋为空

end[i] = NULL;

p->link = NULL;

if(runTime(p->grade)+p->rtime < p->ntime)

{

//把该进程插到下一队列的末尾

p->rtime += runTime(p->grade);

if(p->grade == MAXQUEUENUM)//最后一个队列，仍然插到本队列

addPCBIntoList(MAXQUEUENUM-1);

else//插到下一个队列当中

addPCBIntoList(p->grade++);

return ;

}

else

//说明该进程已经完成

destroy();

return ;

}

}

//作业控制块相关操作

void addJCB(JCB\* newJCB)

//把作业控制块添加到后备作业链当中

{

if(newJCB==NULL)

return ;

if(headJCB==NULL)

{//未有作业

endJCB = headJCB = newJCB;

}

else

{//队列未空，直接插入到队尾

endJCB->next = newJCB;

endJCB = endJCB->next;

}

newJCB = NULL;

}

//为新的作业创建JCB控制块

void createJCB()

{

JCB\* newJCB = getpch(JCB);

//分配存储空间

if(newJCB == NULL)

{

printf("系统内存不够!\n");

exit(-1);

}

printf("----》请输入用户名: ");

scanf("%s",newJCB->userName);

printf("\n----》请输入作业名: ");

scanf("%s",newJCB->jobName);

printf("\n----》请输入作业的到达时间:(如13 20)");

scanf("%d %d",&newJCB->reachTime[0],&newJCB->reachTime[1]);

printf("\n----》请输入运行时间: ");

scanf("%d",&newJCB->runTime);

printf("\n----》请输入所需的主存空间(K): ");

scanf("%d",&newJCB->resourceMainMemory);

printf("\n----》请输入所需的磁带机数: ");

scanf("%d",&newJCB->resourceTapeNum);

newJCB->order = 0;

newJCB->state = 'w';

newJCB->next = NULL;

getchar();

addJCB(newJCB);

//添加到JCB链中

printf("创建作业成功.\n");

jobInvoke();

//作业调度

}

//动态创建并初始化单个作业控制块，并返回其指针

JCB\* initJCB(char\* userName, char\* jobName, char state, int reachTime[], int runTime,

int resourceMainMemory, int resourceTapeNum)

{

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

if(newJCB == NULL)

{

printf("内存空间不够!");

exit(1);

}

strcpy(newJCB->userName,userName);

//用户名复制

strcpy(newJCB->jobName,jobName);

newJCB->state = state;

newJCB->reachTime[0] = reachTime[0];

newJCB->reachTime[1] = reachTime[1];

newJCB->runTime = runTime;

newJCB->resourceMainMemory = resourceMainMemory;

newJCB->resourceTapeNum = resourceTapeNum;

newJCB->next = NULL;

newJCB->order = 0;

return newJCB;

//返回指针

}

//系统自动创建并初始化5个作业控制块

void initJCBLinkList()

{

JCB\* newJCB = NULL;

int reachTime[5][2]={9,0,9,20,9,30,9,35,9,45};

newJCB = initJCB("用户一","作业一",'w',reachTime[0],15,20,2);

addJCB(newJCB);

newJCB = initJCB("用户二","作业二",'w',reachTime[1],18,60,1);

addJCB(newJCB);

newJCB = initJCB("用户三","作业三",'w',reachTime[2],9,45,3);

addJCB(newJCB);

newJCB = initJCB("用户四","作业四",'w',reachTime[3],12,10,2);

addJCB(newJCB);

newJCB = initJCB("用户五","作业五",'w',reachTime[4],6,25,3);

addJCB(newJCB);

}

void dispMenu()

{

system("cls");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\t\t\t 作业调度模拟程序 高胜立 网络二班 3109006516 \n");

printf("++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n");

printf("→\t\t1: 增加作业 \n");

printf("→\t\t2: 查询作业状态 \n");

printf("→\t\t3: 查询已完成作业 \n");

printf("→\t\t4: 查看当前进程 \n");

printf("→\t\t5: 执行 \n");

printf("→\t\t6: 退出 \n");

}

//打印输出作业队列的信息

void dispJCBLinkList()

{

JCB\* jcbTemp = headJCB;

if(jcbTemp == NULL)

{

printf("当前没有作业处于后备队列当中\n");

return ;

}

printf("用户名\t作业名\t状态\t到达时间\t运行时间\t主存\t磁带机\t调用次序\n");

while(jcbTemp)

{

printf("%s\t",jcbTemp->userName);

printf("%s\t",jcbTemp->jobName);

printf("%c\t",jcbTemp->state);

if(jcbTemp->reachTime[1] != 0)

printf("%d:%d\t\t",jcbTemp->reachTime[0],jcbTemp->reachTime[1]);

else

printf("%d:%d0\t\t",jcbTemp->reachTime[0],jcbTemp->reachTime[1]);

printf("%d\t\t",jcbTemp->runTime);

printf("%dK\t",jcbTemp->resourceMainMemory);

printf("%d\t",jcbTemp->resourceTapeNum);

printf("%d\n",jcbTemp->order);

jcbTemp = jcbTemp->next;

}

}

void jobInvoke()

//作业调度核心模块

{

JCB\* temp = headJCB;

while(temp)

{

if(temp->resourceMainMemory<=tatalResourceMainMemory

&& temp->resourceTapeNum<=totalResourceTapeNum && temp->state=='w')

{

//此时可将此作业调进内存

tatalResourceMainMemory -= temp->resourceMainMemory;

totalResourceTapeNum -= temp->resourceTapeNum;

temp->state = 'R';

temp->order = ++choiceOrder;

printf("注意!作业%s已被调入内存运行...\n",temp->jobName);

createPCB(temp);

addPCBIntoList(0);

}

temp = temp->next;

}

}

//把已完成的JCB块添加到已完成作业链当中

void addFinishJob(JCB\* finishJCB)

{

if(headJCBFish==NULL)

headJCBFish = endJCBFish = finishJCB;

else

{

endJCBFish->next = finishJCB;

endJCBFish = endJCBFish->next;

}

}

//打印输出所有已完成作业的信息

void printFinishJCB()

{

JCB\* jcbTemp = headJCBFish;

if(jcbTemp == NULL)

{

printf("当前没有作业已经完成\n");

return ;

}

printf("用户名\t作业名\t状态\t到达时间\t运行时间\t主存\t磁带机\t调用次序\n");

while(jcbTemp)

{

printf("%s\t",jcbTemp->userName);

printf("%s\t",jcbTemp->jobName);

printf("%c\t",jcbTemp->state);

if(jcbTemp->reachTime[1] != 0)

printf("%d:%d\t\t",jcbTemp->reachTime[0],jcbTemp->reachTime[1]);

else

printf("%d:%d0\t\t",jcbTemp->reachTime[0],jcbTemp->reachTime[1]);

printf("%d\t\t",jcbTemp->runTime);

printf("%dK\t",jcbTemp->resourceMainMemory);

printf("%d\t",jcbTemp->resourceTapeNum);

printf("%d\n",jcbTemp->order);

jcbTemp = jcbTemp->next;

}

}

int main()

{

int quit = 0;

dispMenu();

initJCBLinkList(); //作业自动初始化

printf("系统已自动完成5个作业的初始化任务.\n");

jobInvoke();//初始化作业的进程调度

while(!quit)

{

char choice;

printf("请按回车键继续.");

getchar();

dispMenu();

printf("请输入指令: ");

scanf("%c",&choice);

switch(choice)

{

case '1'://添加作业

createJCB();

break;

case '2'://查看作业

dispJCBLinkList();

break;

case '3'://查询已完成作业

printFinishJCB();

break;

case '4':

printf("当前正在运行的进程状态如下:\n");

printAllProcesses();

break;

case '5'://执行进程

run();

break;

case '6'://退出while循环

quit = 1;

break;

default:

printf("无效指令，请重新输入指令.\n");

break;

}

getchar();

}

printf("感谢您对本系统的使用.\n");

printf("按任意键离开.");

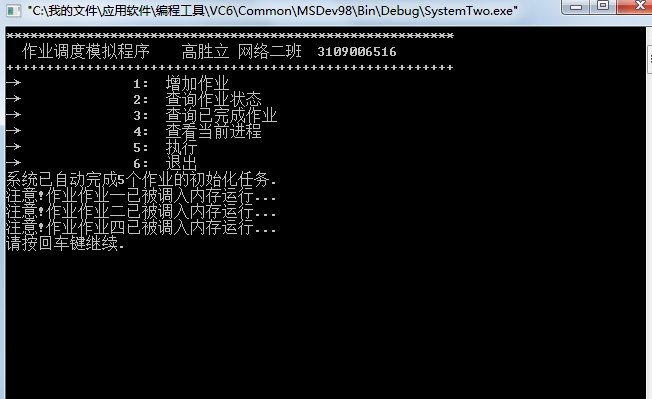
getchar();

return 0;

}

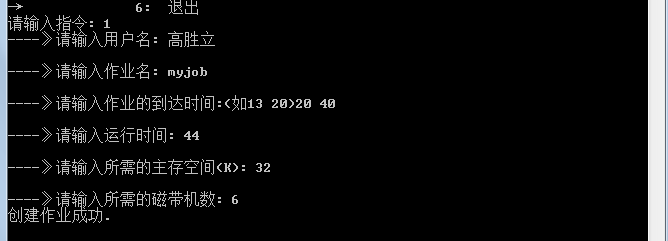
1. **实验结果分析**

* **开始界面，自动初始化五个作业，如下图所示：**



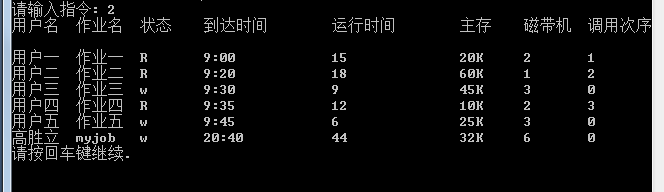
如上图所示，用户可以从中选择操作，进入下一个菜单

* **当选择1时，表示增加作业**

****

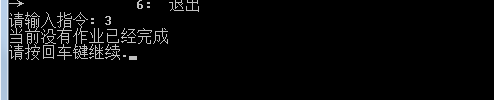
创建作业后提示创建成功，如上图所示。接下来查询当前作业状态，如下一步。

* **选择2，进入查询作业：**

****

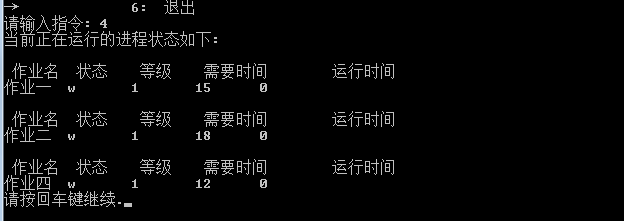
如上图所示，创建的作业已加入了，状态为后备。

* **选择3，查询已完成的作业：**

****

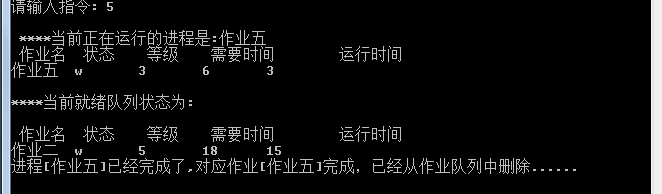
由于没有执行，所以此时查询完成的作业没有，所以给出提示！

* **选择4，查看当前进程**

****

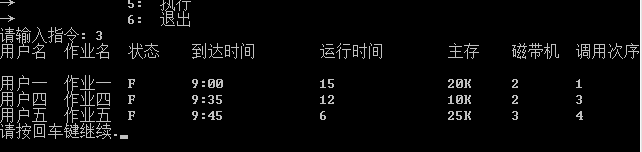
如上图所示，显示当前正在运行的进程！

* **选择5，执行**



如上图是执行了作业五需要的时间次数后显示的结果，作业完成后，从作业队列中删除！

* **此时回看已完成的进程**

****

如上图所示，此时有三个作业完成了

1. **调试总结及心得体会**

* **调试总结**

由于多道操作系统作业调度模拟相对于单道批处理难度大很多，所以本次实验花费了很多的时间去编写代码和调试程序。

由于在做本次实验之前，操作系统的课程没有跟上实验课的脚步，所以要去自学书本上的关于作业调度的知识。

调试程序，在于考虑的细节多和设计的可行性和可操作性。

* 通过本次实验，我对操作系统的知识更加感兴趣了，对作业调度有了一个时刻的理解和认识。获得了很多的关于编程方面的收获。

1. **思考题**

* 写出每种算法的调度策略，最后比较各种算法的优缺点。

答： 先来先服务（FCFS）调度算法是一种最简单的调度算法，该算法既可用于作业调度，也可以用于进程调度。比较有利于长作业（进程），而不利于短作业（进程）。

短作业（进程）优调度算法有利于短作业（进程）的调度，平均周转时间和平均带权周转时间都有明显的改善，但不利于长作业（进程）的调度，肯能导致长作业（进程）长期不被调度。

多级反馈队列调度算法不必事先知道各个进程所需的执行时间，而且还可以满足各种类型进程的需要，是目前被公认的一种较好的进程调度算法。

* 选择调度算法的依据是什么？

答：调度算法的选择是依据需要调度的作业类型，是长作业还是短作业，另外还需要判断是偏重与I/O型还是处理机型。