# 实验一： 作业调度的模拟

## 实验目的

1. 加深对作业概念的理解。
2. 深入了解批处理系统如何组织作业、管理作业和调度作业。
3. 掌握采用银行家算法避免系统死锁的方法。

## 实验预备知识

1. 批处理系统的概念
2. 作业调度算法

## 实验步骤

1. 设计系统总共的作业数。
2. 为每个作业设置到达时间、预计需要服务时间，并设计相应的存储结构。
3. 根据作业调度算法（先来先服务和短作业优先都要做），编写程序，模拟作业调度。
4. 保存调度结果（作业调度次序，每个作业的完成时间、周转时间、带权周转时间，所有作业的平均周转时间、平均带权周转时间）、提交实验结论。

## 实验结论

1. 给出实验源程序（附有详细注释，包括姓名、班级、学号等）

/\*

实验：作业调度的模拟

语言：c++

\*/

#include <iostream>

#include<string>

using namespace std;

//作业的数据结构

struct Job{

char status;//状态

string name;//作业的名字

float arrTime;//到达的时间

float serverTime;//CPU服务的时间

float start;

float final;

float zhouzhuan;

float dqzhouzhan;

};

//先来先服务

void FCFS( Job \*jobs,int jobNum){

//按到来的时间进行排序

for(int i=0;i<jobNum-1;i++){

for(int j=0;j<jobNum-i-1;j++){

if(jobs[j].arrTime>jobs[j+1].arrTime){

Job temp=jobs[j];

jobs[j]=jobs[j+1];

jobs[j+1]=temp;

}

}

}

//设置作业开始，完成，周转，带权时间

jobs[0].start=jobs[0].arrTime;

jobs[0].final=jobs[0].start+jobs[0].serverTime;

jobs[0].zhouzhuan=jobs[0].final-jobs[0].arrTime;

jobs[0].dqzhouzhan=jobs[0].zhouzhuan/jobs[0].serverTime;

for(int i=1;i<jobNum;i++){

if(jobs[i].arrTime<=jobs[i-1].final){

jobs[i].start=jobs[i-1].final;

}

else{

jobs[i].start=jobs[i].arrTime;

}

jobs[i].final=jobs[i].start+jobs[i].serverTime;

jobs[i].zhouzhuan=jobs[i].final-jobs[i].arrTime;

jobs[i].dqzhouzhan=jobs[i].zhouzhuan/jobs[i].serverTime;

}

cout<< "作业调度次序"<<" "<<"作业的名字"<<" "<<"作业的开始时间"<<" "<<"作业的完成时间"<<" "<<"服务的时间"<<" "<<"周转时间"<<" "<<"带权周转时间"<<endl;

float zhouzhuanSum=0;

float dqzhouzhuanSum=0;

for(int i=0;i<jobNum;i++){

cout<< (i+1) <<" "<<jobs[i].name<<" "<<jobs[i].start<<" "<<jobs[i].final<<" "<<jobs[i].serverTime<<" "<<jobs[i].zhouzhuan<<" "<<jobs[i].dqzhouzhan<<endl;

jobs[i].status='1';

zhouzhuanSum += jobs[i].zhouzhuan;

dqzhouzhuanSum += jobs[i].dqzhouzhan;

}

cout<<"平均周转时间:"<<zhouzhuanSum/jobNum<<"平均带权周转时间:"<<dqzhouzhuanSum/jobNum<<endl;

}

//短作业优先,非抢占式

void SJF(Job \*jobs,int jobNum){

//按照服务的时间大小进行排序

for(int i=0;i<jobNum-1;i++){

for(int j=0;j<jobNum-i-1;j++){

if(jobs[j].serverTime>jobs[j+1].serverTime){

Job temp=jobs[j];

jobs[j]=jobs[j+1];

jobs[j+1]=temp;

}

}

}

cout<< "作业调度次序"<<" "<<"作业的名字"<<" "<<"作业的开始时间"<<" "<<"作业的完成时间"<<" "<<"服务的时间"<<" "<<"周转时间"<<" "<<"带权周转时间"<<endl;

float currentTime=0;//当前的时间

float zhouzhuanSum=0;

float dqzhouzhuanSum=0;

for(int i=0;i<jobNum;i++){

for(int j=0;j<5;j++){

if( jobs[j].arrTime<=currentTime && jobs[j].status=='0'){

jobs[j].start=currentTime;

jobs[j].final= jobs[j].start+ jobs[j].serverTime;

jobs[j].zhouzhuan=jobs[j].final-jobs[j].arrTime;

jobs[j].dqzhouzhan=jobs[j].zhouzhuan/jobs[j].serverTime;

currentTime= jobs[j].final;

jobs[j].status='1';

zhouzhuanSum += jobs[j].zhouzhuan;

dqzhouzhuanSum += jobs[j].dqzhouzhan;

cout<< (i+1) <<" "<<jobs[j].name<<" "<<jobs[j].start<<" "<<jobs[j].final<<" "<<jobs[j].serverTime<<" "<<jobs[j].zhouzhuan<<" "<<jobs[j].dqzhouzhan<<endl;

break;

}

}

}

cout<<"平均周转时间:"<<zhouzhuanSum/jobNum<<"平均带权周转时间:"<<dqzhouzhuanSum/jobNum<<endl;

}

int main (void){

int jobNum;

Job \*jobs;

cout<<"输入要运行的作业的个数"<<endl;

cin>>jobNum;

jobs = new Job[jobNum];

//为每个作业设置到达时间、预计需要服务时间

for(int i=0;i<jobNum;i++){

jobs[i].status='0';

cout<<"第"<<(i+1) <<"个作业："<<endl;

cout<<"作业"<<(i+1) <<"名称：";

cin >> jobs[i].name;

cout<<"到达系统的时间:";

cin>>jobs[i].arrTime;

cout<<"服务的时间:";

cin>>jobs[i].serverTime;

}

//选择调度算法

cout<<"使用先来先服务，按1"<<endl;

cout<<"使用短作业优先，按2"<<endl;

int n;

cin>>n;

if(n==1){

FCFS(jobs,jobNum);

}

if(n==2){

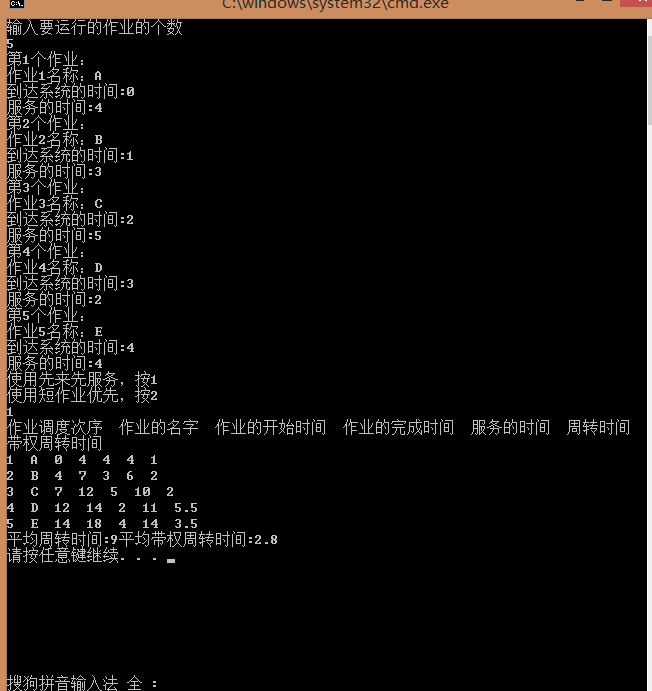
SJF(jobs,jobNum);

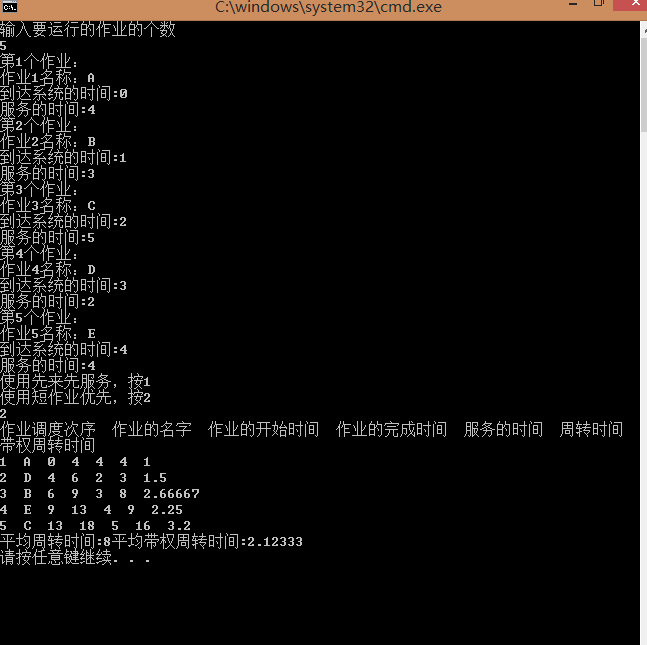
}

return 0;

}

1. 程序运行截图





1. 比较一下两种调度算法的特点

**FCFS算法比较有利于长作业（进程），而不利于短作业（进程）。**

**SJF调度算法的优缺点：**

**优点：有效降低作业的平均等待时间，提高系统吞吐量。**

**缺点：**

**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（进程）会被及时处理。**

**由于作业（进程）的长短含主观因素，不一定能真正做到短作业优先。**