**郑州轻工业大学**

**《操作系统》课程****实验报告**

实验名称：动态高优先权优先调度算法

姓 名： 原彬贺

学 号： 542001020223

地 点： 实验楼302

指导老师： 吴庆岗

专业班级： 计算机科学与技术20-02班

实验成绩：

1. 实验目的：

编写程序，实现动态高优先权优先调度算法，模拟进程调度过程，加深对进程调度的理解。

二、实验内容：

模拟实现动态高优先权优先的调度（若数值越大优先权越高，每运行一个时间单位优先权-n，若数值越小优先权越高，每运行一个时间单位优先权+n）。实验要求：

1、设计进程控制块(PCB)的数据结构

–应包含实验必须的数据项，如进程名，进程的到达时间，服务时间，初始优先权，进程状态（W——等待,R——运行,F——完成），进程间的链接指针，以及实验者认为有必要的其他数据项。

2、实现排序算法（将进程排队）

–对就绪状态的进程按照优先权排序。优先权相同时进入等待队列时间早的进程在前，注意考虑到达时间。

3、实现调度过程模拟

–每个进程用一个PCB表示，按步骤2方法进程排队；每次从等待队列队首调度优先权最高的进程执行，状态变化；在执行一个时间单位后优先权变化，服务时间变化，状态变化；当服务时间为0时，状态变为F；撤销状态为F的进程。

4、实现结果输出

– 输出进程状态表，展示调度过程

* + 初始进程状态（未调度时）
  + 每次调度后的进程状态

5、实验要求

（1）测试数据可以随机输入或从文件中读入；

（2）最终能够计算每一个进程的周转时间、带权周转时间

三、实验代码

#include "stdio.h"

#include<iostream>

#include <vector>

#include <ctime>

#include <list>

#include <algorithm>

#include <iterator>

using namespace std;

enum tcbstatus { wait, done, doing };

class TCB {

private:

int id;

int startTime;

int doneTIme;

int serverTime;

int creatTime;

int waitTime;

tcbstatus status;

float priority;

public:

TCB(int ID, int ServerTime, int CreattTiem) :id(ID), serverTime(ServerTime), creatTime(CreattTiem) {

this->waitTime = 0;

this->priority = FLT\_MAX;

this->status = wait;

}

int getid()

{

return id;

}

void setStartTime(int time)

{

startTime = time;

}

tcbstatus getstatus()

{

return this->status;

}

int getcreatTiem()

{

return creatTime;

}

int getstartTiem()

{

return startTime;

}

int getdoneTime()

{

return doneTIme;

}

int getserverTime()

{

return serverTime;

}

bool setdoneTime()

{

doneTIme = std::time(0);

}

bool cmpcreat(TCB const\* peer)

{

if (this->creatTime <= peer->creatTime)

{

return true;

}

else return false;

}

bool cmppr(TCB \* peer)

{

if (this->getpriority() <= peer->getpriority())

{

return true;

}

else return false;

}

bool cmpserver(TCB const\* peer)

{

if (this->serverTime <= peer->serverTime)

{

return true;

}

else

{

return false;

}

}

int setrun(int t)

{

serverTime = serverTime - t;

if (serverTime <= 0)

{

status = done;

t +=serverTime;

cout << " done ";

}

else {

status = wait;

cout << " not done";

this->priority = FLT\_MIN;

}

waitTime = 0;

cout << " run:" << t << " time" << endl;

return t;

}

float getpriority()

{

return priority;

}

void setwait(int t)

{

waitTime += t;

priority = float((waitTime + serverTime)) / serverTime;

}

bool operator > (TCB const\* tcb)

{

if (this->priority >= tcb->priority)

{

return true;

}

else

{

return false;

}

}

};

class my\_great{

public:

bool operator()(TCB\* t,TCB\* s)

{

if (t->getpriority() >= s->getpriority())

{

return true;

}

else

{

return false;

}

}

};

class my\_greatti {

public:

bool operator()(TCB\* t, TCB\* s)

{

if (t->getserverTime() >= s->getserverTime())

{

return true;

}

else

{

return false;

}

}

};

class TCBworks {

private:

list<TCB\*> tcbworks;

int status;

public:

TCBworks(int stuts) :status(stuts)

{

}

void setStatus(int sta)

{

status = sta;

return;

}

int startrun(int t)

{

if (tcbworks.size() == 0)

{

return 0;

}

else {

auto itr = tcbworks.begin();

cout << "id = " << (\*itr)->getid() << " starttime: = " << (\*itr)->getcreatTiem() << " srevertime : = " << (\*itr)->getserverTime() << " priority : = " << (\*itr)->getpriority() ;

t =(\*itr)->setrun(t);

if ((\*itr)->getstatus() == done)

{

tcbworks.erase(itr);

itr = tcbworks.begin();

}

else {

itr++;

}

for (; itr != tcbworks.end(); itr++)

{

(\*itr)->setwait(t);

}

tcbworks.sort([](TCB\* t, TCB\* s) {

if (t->getpriority() < s->getpriority())

return true;

else

return false;

});

}

this->show();

return t;

}

void show()

{

int j = 0;

tcbworks.sort([]( TCB\* t, TCB\* s) {

if (t->getserverTime() < s->getserverTime())

return true;

else

return false;

});

list<TCB\*>::iterator itr = tcbworks.begin();

for (; itr != tcbworks.end(); itr++)

{

if (status == 0)

{

cout << "id = " << (\*itr)->getid() << " starttime: = " << (\*itr)->getcreatTiem() << " srevertime : = " << (\*itr)->getserverTime() << " priority : = " << (\*itr)->getpriority() << endl;;

}

else

{

cout << "id = " << (\*itr)->getid() << " starttime: = " << (\*itr)->getcreatTiem() << " srevertime : = " << (\*itr)->getserverTime() << " priority : = " << (\*itr)->getpriority();

}

j++;

}

}

void appendTCB(TCB\* tcb)

{

auto it = tcbworks.begin();

switch (status)

{

case 0: {

for (; it != tcbworks.end(); it++)

{

if (tcb->cmpcreat(\*(it)))

{

tcbworks.insert(it, tcb);

break;

}

}

if (it == tcbworks.end())

{

tcbworks.insert(it, tcb);

}

break;

}

case 1:

{

for (; it != tcbworks.end(); it++)

{

if (!tcb->cmpserver((\*it)))

{

tcbworks.insert(it, tcb);

break;

}

}

if (it == tcbworks.end())

{

tcbworks.insert(it, tcb);

}

break;

}

case 2:

{

for (; it != tcbworks.end(); it++)

{

if (tcb->cmppr((\*it)))

{

tcbworks.insert(it, tcb);

break;

}

}

if (it == tcbworks.end())

{

tcbworks.insert(it, tcb);

}

break;

}

}

}

};

int main()

{

int nums[10] = { 10,3,1,5,6,1,4,6,2,7 };

int time[10] = { 1,3,4,2,5,2,6,7,3,7 };

TCBworks\* works = new TCBworks(0);

for (int i = 0; i < 10; i++)

{

TCB\* jcb = new TCB(i, nums[i], time[i]);

works->appendTCB(jcb);

}

int runtime = 2;

int ti = 0;

//works->show();

while (1)

{

cout << "the " << ti << " time: ";

int t = works->startrun(runtime);

if (t == 0)

{

cout << "all done";

break;

}

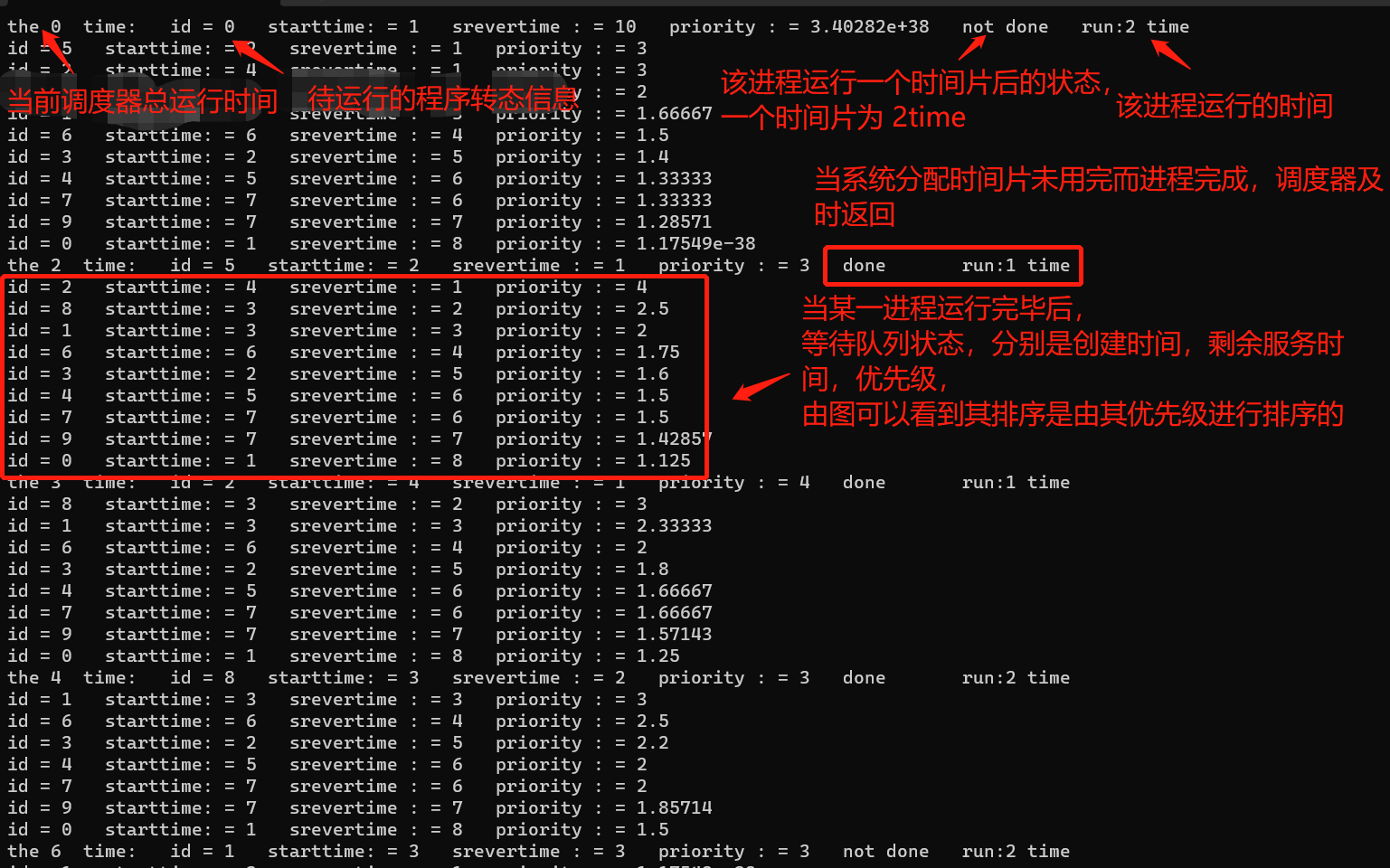
ti += t;

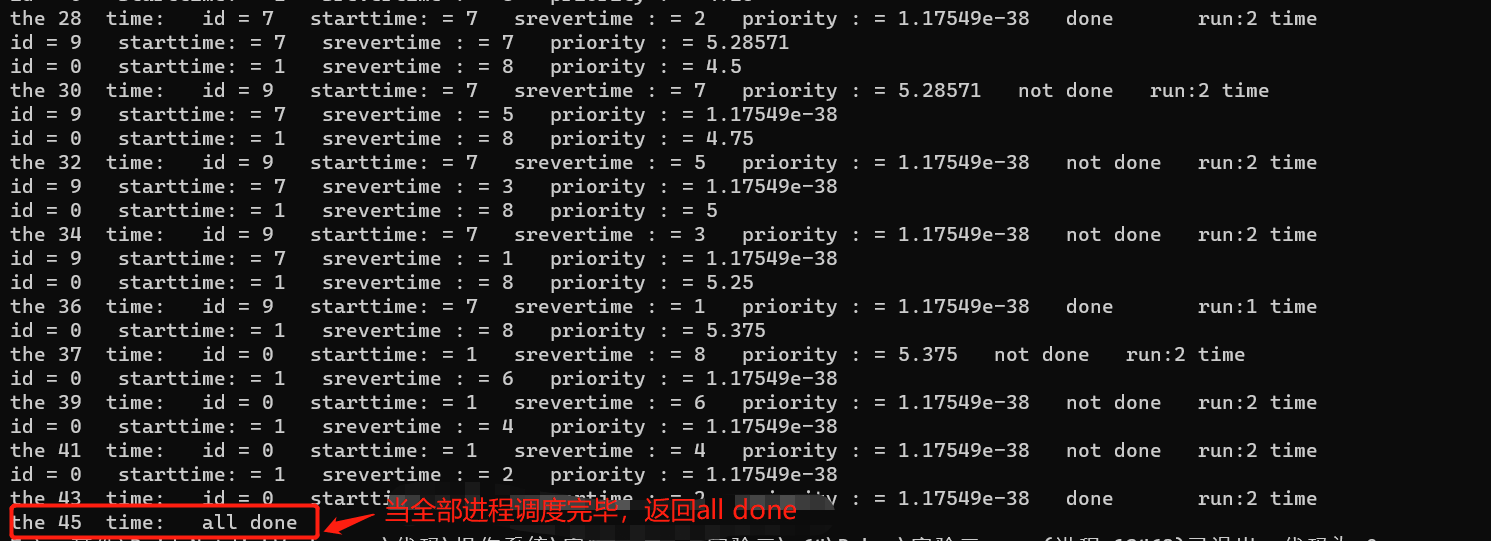
}

return 0;

}

四、实验结果





五、实验总结

在本次实验，我选择了一些简单的任务作为实验对象，将它们按照优先级和时间要求进行排序，并将它们存储在一个数组中。然后，我们使用动态规划的方法来计算每个任务的最小完成时间，并将它们按照优先级和时间要求进行排序。最后，我将排好序的任务输出到调度器中进行调度。通过对运行结果进行查看，可以看到其能能尽量均衡的使每个进程在一段时间内都有机会被调度到，避免了某些进程始终无法被调度到的问题。这次实验使我收获很大，使我更加深入的了解到了动态优先级算法的好处，同时也让我在编程中有了一种新的思维方式。

|  |
| --- |
| **评语**（不要删除，指导老师填写评语，放置在实验报告后面单独成页） |
|  |