

**计算机与信息 学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | 操作系统实验 | | | | |
| 实验编号： | 实验2 | | | | |
| 实验名称： | 多级反馈队列调度算法 | | | | |
| 实验人员： | 学号 | 18111207248 | | | |
| 姓名 | 吴钰 | | | |
| 班级 | 18级计算机科学与技术创新班 | | | |
| 实验日期： | 2020.5.1 | | | | |
| 实验室： |  | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

# 一、实验目的

# 写一个程序描述进程调度过程，加深理解多级反馈队列进程调度算法

# 二、实验要求

# 1) 参数设计，包括级数、每级（队列）时间片大小

# 2) 进程数及每个进程基本信息输入

# 3) 根据多级反馈队列调度算法画出进程调度结果

# 三、设计

【定义所有抽象数据类型，自定义函数的功能详细描述（伪代码表示），以及主程序的流程图。】

#include <iostream>

#include <algorithm>

#include <iomanip>

#include <vector>

#include <queue>

using namespace std;

typedef struct PCB

{

int ID; //标识符

int ComeTime; //到达时间

int ServerTime; //服务时间

int FinishTime; //完成时间

int TurnoverTime; //周转时间

double WeightedTurnoverTime; //带权周转时间

}PCB;

const int QueueNum = 3;//就绪队列长度

int timeslice[QueueNum];//第一个时间片

void create(vector<PCB> &PCBList, int xlice[]);//输入时间片大小，作业信息

bool Cmpp(const PCB &p1, const PCB &p2);//比较到达时间的先后

void MFQ(vector<PCB> &PCBList, int timeslice[]);//多级反馈队列调度

void output(vector<PCB> &PCBList);//显示结果

int main()

{

vector<PCB> PCBList;

create(PCBList, timeslice);

MFQ(PCBList, timeslice);//多级反馈队列调度

output(PCBList);//显示结果

return 0;

}

void create(vector<PCB> &PCBList, int xlice[])

{

int num;

cout << "输入3个时间片大小: "<<endl;

for(int i=0;i<3;i++){

cin >> timeslice[i];

}

cout<<"输入进程数："<<endl;

cin>>num;

PCB temp[20];

for(i=0;i<num;i++){

cout << "输入标识符 "<<"到达时间 "<<"服务时间 "<<endl;

cin >> temp[i].ID>> temp[i].ComeTime>> temp[i].ServerTime;

temp[i].FinishTime = 0; //暂时存放运行了多少时间，来判断此作业是否运行结束

PCBList.push\_back(temp[i]);

}

}

bool Cmpp(const PCB &p1, const PCB &p2)//比较到达时间的先后

{

return p1.ComeTime < p2.ComeTime;

}

void MFQ(vector<PCB> &PCBList, int timeslice[])//多级反馈队列调度算法

{

sort(PCBList.begin(), PCBList.end(), Cmpp); //按到达时间排序

vector<PCB> result;//用于保存结果

int BeginTime = (\*PCBList.begin()).ComeTime; //第一个作业开始时间

queue<PCB> Ready[QueueNum];//设置3个就绪队列

Ready[0].push(\*PCBList.begin());

PCBList.erase(PCBList.begin());

cout<<"当前时刻:"<<BeginTime<<" 当前就绪队列:"<<0<<endl;

cout<<"第一个进程进入就绪队列0"<<endl;

cout<<endl;

while (!PCBList.empty())//进程数组不空

{

int i; //了防止前面的进程运行完了，后面的进程还没到，造成死循环

bool flag=false;

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

{

if(!Ready[i].empty())//就绪队列不为空

{

flag = true;

break;

}

}

if(!flag)//就绪队列不为空

{

Ready[0].push(\*PCBList.begin());

PCBList.erase(PCBList.begin());

BeginTime=Ready[0].front().ComeTime;

}

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

{

if (i!=QueueNum-1) //不是最后一个队列

{

while(!Ready[i].empty()) //当前队列不空

{

if (!PCBList.empty()&&BeginTime>=(\*PCBList.begin()).ComeTime) //有新作业到达，加入就绪队列，转到第一队列

{

cout<<"当前时刻:"<<BeginTime<<endl;

cout<<"新进程"<<(\*PCBList.begin()).ID<<"到达，将其放在第一队列尾部"<<endl;

cout<<endl;

Ready[0].push(\*PCBList.begin());

PCBList.erase(PCBList.begin());

i=0;

continue;

}

if (Ready[i].front().FinishTime + timeslice[i]<Ready[i].front().ServerTime)//时间片用完没运行完,加入下一队列队尾

{

cout<<"当前时刻:"<<BeginTime+ timeslice[i]<<" 当前就绪队列:"<<i<<endl;

cout<<"当前进程"<<Ready[i].front().ID<<"在时间片内没有运行完，加入下一个队列尾部！"<<endl;

cout<<endl;

Ready[i].front().FinishTime+=timeslice[i] ;

Ready[i+1].push(Ready[i].front());//加入下一个队列

Ready[i].pop();

BeginTime+= timeslice[i] ;

}

else//此作业运行完

{

BeginTime += Ready[i].front().ServerTime-Ready[i].front().FinishTime;

Ready[i].front().FinishTime = BeginTime;

Ready[i].front().TurnoverTime = Ready[i].front().FinishTime - Ready[i].front().ComeTime;

Ready[i].front().WeightedTurnoverTime = (double)Ready[i].front().TurnoverTime / Ready[i].front().ServerTime;

cout<<"当前时刻:"<<BeginTime<<" 当前就绪队列:"<<i<<endl;

cout<<"当前进程"<< Ready[i].front().ID<<"在时间片内运行完！"<<endl;

cout<<endl;

result.push\_back(Ready[i].front());//从就绪队列中移除作业

Ready[i].pop();

}

}

}

else

{

while (!Ready[i].empty())

{

if (!PCBList.empty() && BeginTime >= (\*PCBList.begin()).ComeTime)//有新作业到达，加入就绪队列，转到第一队列

{

cout<<"当前时刻:"<<BeginTime<<endl;

cout<<"新进程"<<(\*PCBList.begin()).ID<<"到达，将其放在最后队列尾部"<<endl;

cout<<endl;

Ready[0].push(\*PCBList.begin());

PCBList.erase(PCBList.begin());

i=-1;

break;

}

if(Ready[i].front().FinishTime + timeslice[i] < Ready[i].front().ServerTime)//时间片用完没运行完,加入队尾

{ cout<<"当前时刻:"<<BeginTime+ timeslice[i]<<" 当前就绪队列:"<<i<<endl;

cout<<"当前进程"<<Ready[i].front().ID<<"在时间片内没有运行完，加入该队列尾部！"<<endl;

cout<<endl;

Ready[i].front().FinishTime += timeslice[i] ;

Ready[i].push(Ready[i].front());

Ready[i].pop();

BeginTime += timeslice[i] ;

}

else//此作业运行完

{

BeginTime+=Ready[i].front().ServerTime - Ready[i].front().FinishTime;

Ready[i].front().FinishTime=BeginTime;

Ready[i].front().TurnoverTime=Ready[i].front().FinishTime - Ready[i].front().ComeTime;

Ready[i].front().WeightedTurnoverTime=(double)Ready[i].front().TurnoverTime / Ready[i].front().ServerTime;

cout<<"当前时刻:"<<BeginTime<<" 当前就绪队列:"<<i<<endl;

cout<<"当前进程"<< Ready[i].front().ID<<"在时间片内运行完！"<<endl;

cout<<endl;

result.push\_back(Ready[i].front());//从就绪队列中移除作业

Ready[i].pop();

}

}

}

}

}

//按ComeTime升序排序，便于显示结果

PCBList=result;

sort(PCBList.begin(),PCBList.end(),Cmpp);

}

//显示结果

void output(vector<PCB> &PCBList)

{

int SumTurnoverTime=0;

double SumWeightedTurnoverTime=0;

cout<<"标识符 "<<"达到时间 "<<"服务时间 "<<"完成时间 "<<"周转时间 "<<"带权周转时间" <<endl;

for (vector<PCB>::iterator it = PCBList.begin(); it < PCBList.end(); ++it){

cout<<(\*it).ID<<" "<<(\*it).ComeTime<<" "<< (\*it).ServerTime<<" "<<(\*it).FinishTime<<" "<<(\*it).TurnoverTime<<" "<< (\*it).WeightedTurnoverTime<<endl;

SumTurnoverTime+=(\*it).TurnoverTime;

SumWeightedTurnoverTime+=(\*it).WeightedTurnoverTime;

}

cout << "平均周转时间: " << (double)SumTurnoverTime/PCBList.size() << endl;

cout << "平均带权周转时间: "<<SumWeightedTurnoverTime/PCBList.size() << endl;

}

# 四、调试分析和思考

【(1) 调试过程中遇到的问题是如何解决的以及对设计与实现中关键点的回顾讨论和分析；(2) 算法的时空分析(包括基本操作和其他算法的时间复杂度和空间复杂度的分析)和改进设想；(3) 经验和体会等。】

答：1、思想：设置多个就绪队列。在系统中设置多个就绪队列，并为每个队列赋予不同的优先。第一个队列的优先级最高，第二个次之，其余队列的优先级逐个降低。该算法为不同列中的进程所赋予的执行时间片的大小也各不相同，在优先级愈高的队列中，其时间片愈小。每个队列都采用FCFS算法。当新进程进入内存后，首先将它放入第一队列的末尾，按FCFS原则等待调度。当轮到该进程执行时，如它能在该时间片内完成，便可撤离系统。否则，即它在一个时间片结束时尚未完成，调度程序将其转入第二队列的末尾等待调度;如果它在第二队列中运行个时间片后仍未完成， 再依次将它放入第三队列...依此类推。当进程最后被降到第n队列后，在第n队列中便采取按RR方式运行。按队列优先级调度。调度程序首先调度最高优先级队列中的诸进程运行，仅当第一队列空闲时才调度第二队列中的进程运行;仅当第1到(i-1)所有队列均空时，才会调度第i队列中的进程运行。如果处理机正在第i队列中为某进程服务时又有新进程进入任一优先级较高的队列，此时须立即把正在运行的进程放回到第i队列的末尾，而把处理机分配给新到的高优先级进程。

2、实验中遇到的问题：整个进程断断续续并且有很多进程和变量不容易计算等待时间。

# 五、测试数据与结果

【列出你的测试结果，明确输入和输出数据。测试数据应该完整和严格，可以直接贴结果图。】

# C:\Users\wuyu\Documents\Tencent Files\1124274751\FileRecv\MobileFile\Image\H1K)%3PVHPV{`$L3T)QV43O.png

