1. 需求分析和任务定义
   1. 需求分析

处理机调度是OS中至关重要的部分，在操作系统中根据不同的调度层次，设计了不同的调度算法。其目的就是使CPU提高系统资源的利用率，使作业或进程在等待和执行时间的分配更加优化，满足不同作业用户需求的要求。

本次设计需要我们利用C++编程语言，实现四种CPU调度算法的模拟，本设计以进程调度为例子。

先来先服务（FCFS）调度算法，它使最简单的调度算法，系统按照作业或进程到达的先后次序来进行调度，选择一个最先进入等待队列的进程投入运行，直到该运行完成或发生阻塞后，才将处理机资源分配给其他资源。

优先级调度算法PSA，基于作业的紧迫程度，由算法赋予每个作业相应的优先级，保证紧迫性作业优先运行。

短作业优先调度算法SJF，与优先级算法相类似，但是按照作业的运行时间长短来计算优先级。

时间片轮转调度算法，采用公平的处理机分配方式，就绪队列每个进程每次运行同样的时间片时间，没执行完程序中断，等待下一个时间片，执行完等待时间片完全消耗，保证每个程序都在一个确定的时间段内获得一次CPU执行。



Typedef struct PCB

{

char Pname;//进程名称

int ArriveTime;//到达时间

int StartTime;//开始时间

int FinishTime;//完成时间

int NeedTime;//需要服务时间

int TdTime;//周转时间

int Priority;//优先级

int AverageTdTime;//平均周转时间

int AverageWaitTime;//平均等待时间

};

PCB::PCB()

{

Pname=0;

ArriveTime=0;

StartTime=0;

FinishTime=0;

TdTime=0;

Priority=0;

AverageTdTime=0;

AverageWaitTime=0;

}

//排序

void sort(PCB\*pcb,int n)

{

int i,j;

for(i=0;i<n-1;i++)

{

for(j=i+1;j<n;j++)

{

if(pcb[i].ArriveTime< pcb [j].ArriveTime){}

else if(pcb [i].ArriveTime== pcb [j].ArriveTime)

{

if(pcb [i].NeedTime<= pcb [j].NeedTime){;}

else

{

int temp;

temp= pcb [i];

pcb [i]= pcb [j];

pcb [j]=temp;

}

}

else if(p[i].ArriveTime>p[j].ArriveTime)

{

int temp;

temp=p[i];

p[i]=p[j];

p[j]=temp;

}

}

}

}

//运行进程

int running (PCB\*p,int n)

{

int r;

for(r=0;r<n;r++)

{

if(r==0)

{

p[r].StartTime=p[r].ArriveTime;

p[r].FinishTime=p[r].ArriveTime+p[r].NeedTime;

}

else

{

p[r].StartTime=p[r-1].FinishTime;

p[r].FinishTime=p[r].StartTime+p[r].NeedTime;

}

}

//计算周转时间

for(r=0;r<n;r++)

{

p[r].TdTime=p[r].FinishTime-p[r].ArriveTime;

}

return 0;

}

//打印进程排序就绪队伍表

void show(PCB\*p,int n)

{}

void FSFC()

{

cout<<”输入调度进程的数量:”;

cin>>n;

//如何得到早先初始化的数据

PCB\*p;

sort(p,n);

cout<<”打印就绪列表:”<<endl;

show(p,n);

running(p,n);

cout<<”进程完成信息如下:”<<endl;

show(p,n);

}