void sortOfArriveTime(PCB \*p, int n)

{ //按到达时间对进程排序

PCB \*t = new PCB;

PCB \*q = new PCB;

PCB \*m = new PCB;

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

{ //冒泡循环

q = p->next; // q指向链表中的第一个进程

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

{

m = q->next;

if (q->arriveTime > m->arriveTime)

{ //结点信息进行交换

t->pName = q->pName;

t->arriveTime = q->arriveTime;

t->serviceTime = q->serviceTime;

t->estimatedRunningtime = q->estimatedRunningtime;

q->pName = m->pName;

q->arriveTime = m->arriveTime;

q->serviceTime = m->serviceTime;

q->estimatedRunningtime = m->estimatedRunningtime;

m->pName = t->pName;

m->arriveTime = t->arriveTime;

m->serviceTime = t->serviceTime;

m->estimatedRunningtime = t->estimatedRunningtime;

}

q = q->next;

}

}

t = NULL;

m = NULL;

q = NULL;

delete t;

delete m;

delete q;

}

void runProcess(PCB \*p, int n)

{ //运行进程

PCB \*q = new PCB;

PCB \*m = new PCB;

PCB \*s = new PCB;

int a = n;

sortOfArriveTime(p, n);

q = p;

m = p->next;

int currentTime = 0; //当前时间

int i = 0;

int number = 0;

while (true)

{

currentTime++;

if (i == 0 && m->arriveTime > currentTime) //首次运行进程

continue;

number = 0;

while (m && m->state == 'C' || m && m->arriveTime > currentTime)

{ //寻找应该访问的进程

number++;

q = m;

m = m->next;

if (m == NULL)

{

q = p;

m = p->next;

}

if (number > n)

break;

}

if (number > n) //所有进程都不能进行访问

continue;

m->estimatedRunningtime--;

m->finishTime = currentTime + 1;

if (m->estimatedRunningtime == 0)

m->state = 'C';

// 输出表头信息（已省略）

s = p->next;

while (s)

{ //输出就绪队列

if (s->estimatedRunningtime != 0)

{// 输出信息

}

s = s->next;

}

cout << endl

<< endl

<< endl;

q = m;

m = m->next; // q、m指针后移

if (m == NULL)

{ //回到链表头部

q = p;

m = p->next;

}

s = p->next;

while (s && s->state == 'C')

s = s->next;

if (s == NULL) //若所有进程已完成，则退出循环

break;

i++;

}

q = p;

m = p->next;

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

{ //计算开始时间、周转时间、带权周转时间

if (i == 0)

{

m->startTime = m->arriveTime;

m->turnaroundTime = m->finishTime - m->arriveTime;

m->weightedTuraroundTime = m->turnaroundTime \* 1.0 / m->serviceTime;

}

else

{

m->startTime = q->startTime + 1 > m->arriveTime ? q->startTime + 1 : m->arriveTime;

m->turnaroundTime = m->finishTime - m->arriveTime;

m->weightedTuraroundTime = m->turnaroundTime \* 1.0 / m->serviceTime;

}

m = m->next;

}

q = NULL;

m = NULL;

s = NULL;

delete q;

delete m;

delete s;

cout << endl;

}





