

操作系统实验报告



**实验[ 1 ]： 处理器调度**

**姓 名：**  **於俊涛**

**学 号： 202007020625**

**专业班级： 计算机203**

**系 别： 计算机系**

**学 院： 电子信息与人工智能学院**

## 实验1 处理器调度 预习报告2

**一、实验内容**

按优先数调度算法实现处理器调度。

**二、实验目的**

在采用多道程序设计的系统中，往往有若干个进程同时处于就绪状态。当就绪进程个数大于处理器数时，就必须依照某种策略来决定哪些进程优先占用处理器。本实验模拟在单处理器情况下的处理器调度，帮助学生加深了解处理器调度的工作。

**三、实验原理**

**设计一个按优先数调度算法实现处理器调度的程序。**

(1) 假定系统有五个进程，每一个进程用一个进程控制块PCB来代表，进程控制块的格式为：

|  |
| --- |
| 进程名 |
| 指针 |
| 要求运行时间 |
| 优先数 |
| 状态 |

其中，进程名——作为进程的标识，假设五个进程的进程名分别为P1，P2，P3，P4，P5。

指针——按优先数的大小把五个进程连成队列，用指针指出下一个进程的进程控制块的首地址，最后一个进程中的指针为“0”。

要求运行时间——假设进程需要运行的单位时间数。

优先数——赋予进程的优先数，调度时总是选取优先数大的进程先执行。

状态——可假设有两种状态，“就绪”状态和“结束”状态。五个进程的初始状态都为“就绪”，用“R”表示，当一个进程运行结束后，它的状态为“结束”，用“E”表示。

(2) 在每次运行你所设计的处理器调度程序之前，为每个进程任意确定它的“优先数”和“要求运行时间”。

(3) 为了调度方便，把五个进程按给定的优先数从大到小连成队列。用一单元指出队首进程，用指针指出队列的连接情况。例：

队首标志

K2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| K1 | P1 | K2 | P2 | K3 | P3 | K4 | P4 | K5 | P5 |
|  | 0 |  | K4 |  | K5 |  | K3 |  | K1 |
|  | 2 |  | 3 |  | 1 |  | 2 |  | 4 |
|  | 1 |  | 5 |  | 3 |  | 4 |  | 2 |
|  | R |  | R |  | R |  | R |  | R |
|  | PCB1 |  | PCB2 |  | PCB3 |  | PCB4 |  | PCB5 |

(4) 处理器调度总是选队首进程运行。采用动态改变优先数的办法，进程每运行一次优先数就减“1”。由于本实验是模拟处理器调度，所以，对被选中的进程并不实际的启动运行，而是执行：

优先数-1

要求运行时间-1

来模拟进程的一次运行。

提醒注意的是：在实际的系统中，当一个进程被选中运行时，必须恢复进程的现场，让它占有处理器运行，直到出现等待事件或运行结束。在这里省去了这些工作。

(5) 进程运行一次后，若要求运行时间≠0，则再将它加入队列（按优先数大小插入，且置队首标志）；若要求运行时间=0，则把它的状态修改成“结束”（E），且退出队列。

(6) 若“就绪”状态的进程队列不为空，则重复上面（4）和（5）的步骤，直到所有进程都成为“结束”状态。

(7) 在所设计的程序中应有显示或打印语句，能显示或打印每次被选中进程的进程名以及运行一次后进程队列的变化。

(8) 为五个进程任意确定一组“优先数”和“要求运行时间”，启动所设计的处理器调度程序，显示或打印逐次被选中进程的进程名以及进程控制块的动态变化过程。

## 实验1 处理器调度

**一、实验内容**

按优先数调度算法实现处理器调度。

**二、实验目的**

在采用多道程序设计的系统中，往往有若干个进程同时处于就绪状态。当就绪进程个数大于处理器数时，就必须依照某种策略来决定哪些进程优先占用处理器。本实验模拟在单处理器情况下的处理器调度，帮助学生加深了解处理器调度的工作。

**三、实验原理**

**设计一个按优先数调度算法实现处理器调度的程序。**

(1) 假定系统有五个进程，每一个进程用一个进程控制块PCB来代表，进程控制块的格式为：

|  |
| --- |
| 进程名 |
| 指针 |
| 要求运行时间 |
| 优先数 |
| 状态 |

其中，进程名——作为进程的标识，假设五个进程的进程名分别为P1，P2，P3，P4，P5。

指针——按优先数的大小把五个进程连成队列，用指针指出下一个进程的进程控制块的首地址，最后一个进程中的指针为“0”。

要求运行时间——假设进程需要运行的单位时间数。

优先数——赋予进程的优先数，调度时总是选取优先数大的进程先执行。

状态——可假设有两种状态，“就绪”状态和“结束”状态。五个进程的初始状态都为“就绪”，用“R”表示，当一个进程运行结束后，它的状态为“结束”，用“E”表示。

(2) 在每次运行你所设计的处理器调度程序之前，为每个进程任意确定它的“优先数”和“要求运行时间”。

(3) 为了调度方便，把五个进程按给定的优先数从大到小连成队列。用一单元指出队首进程，用指针指出队列的连接情况。例：

队首标志

K2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| K1 | P1 | K2 | P2 | K3 | P3 | K4 | P4 | K5 | P5 |
|  | 0 |  | K4 |  | K5 |  | K3 |  | K1 |
|  | 2 |  | 3 |  | 1 |  | 2 |  | 4 |
|  | 1 |  | 5 |  | 3 |  | 4 |  | 2 |
|  | R |  | R |  | R |  | R |  | R |
|  | PCB1 |  | PCB2 |  | PCB3 |  | PCB4 |  | PCB5 |

(4) 处理器调度总是选队首进程运行。采用动态改变优先数的办法，进程每运行一次优先数就减“1”。由于本实验是模拟处理器调度，所以，对被选中的进程并不实际的启动运行，而是执行：

优先数-1

要求运行时间-1

来模拟进程的一次运行。

提醒注意的是：在实际的系统中，当一个进程被选中运行时，必须恢复进程的现场，让它占有处理器运行，直到出现等待事件或运行结束。在这里省去了这些工作。

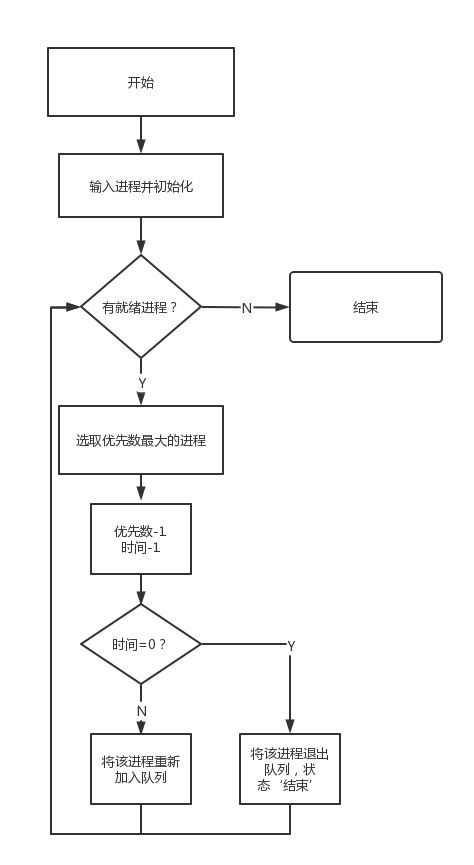
(5) 进程运行一次后，若要求运行时间≠0，则再将它加入队列（按优先数大小插入，且置队首标志）；若要求运行时间=0，则把它的状态修改成“结束”（E），且退出队列。

(6) 若“就绪”状态的进程队列不为空，则重复上面（4）和（5）的步骤，直到所有进程都成为“结束”状态。

(7) 在所设计的程序中应有显示或打印语句，能显示或打印每次被选中进程的进程名以及运行一次后进程队列的变化。

(8) 为五个进程任意确定一组“优先数”和“要求运行时间”，启动所设计的处理器调度程序，显示或打印逐次被选中进程的进程名以及进程控制块的动态变化过程。

**四、算法流程图**



**五、源程序及注释**

数据结构及符号说明：

struct PCB /\*进程控制块结构体,包含标识符、优先数、运行时间、状态、前后指针\*/

{

char name[10]; /\*进程标识符\*/

int run\_time; /\*进程运行时间\*/

int prior\_num; /\*进程优先数\*/

char status; /\*进程状态：R-就绪,E-结束\*/

struct PCB \*pre; /\*指向后一进程的指针\*/

struct PCB \*next; /\*指向后一进程的指针\*/

};

源程序：

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

int num = 5; /\*假定进程数为5\*/

struct PCB\*head; /\*进程链表的头指针\*/

struct PCB\*tail; /\*进程链表的尾指针\*/

int global\_time; /\*定义全局时间\*/

struct PCB\*PCBinit(struct PCB \*q); /\*初始化进程链表\*/

struct PCB\*init(struct PCB \*p, int i); /\*初始化进程\*/

void sort(struct PCB \*phead); /\*冒泡排序链表\*/

void exchange(struct PCB \*p, struct PCB \*max); /\*交换相邻两个进程的指针\*/

void run(struct PCB \*p); /\*模拟运行进程\*/

void showinfor(struct PCB \*head); /\*输出进程信息\*/

void check\_runtime(struct PCB \*p);/\*判断运行时间是否为0\*/

struct PCB /\*进程控制块结构体,包含标识符、优先数、运行时间、状态、前后指针\*/

{

char name[10]; /\*进程标识符\*/

int run\_time; /\*进程运行时间\*/

int prior\_num; /\*进程优先数\*/

char status; /\*进程状态：R-就绪,E-结束\*/

struct PCB \*pre; /\*指向后一进程的指针\*/

struct PCB \*next; /\*指向后一进程的指针\*/

};

//初始化进程

struct PCB\*init(struct PCB \*p, int i){

p->name[0] = 'P';

p->name[1] = 'C';

p->name[2] = 'B';

p->name[3] = i+1+'0';

//为进程指定运行时间

printf("进程 %s\n",p->name);

printf("请确定该进程的运行时间：");

scanf("%d", &p->run\_time);

//为进程指定优先数

printf("请确定该进程的优先数：");

scanf("%d",&p->prior\_num);

printf("\n");

//初始化进程状态为就绪

p->status = 'R';

//初始化指向后一进程的指针为空

p->next = NULL;

return p;

}

//初始化进程链表

struct PCB\*PCBinit(struct PCB \*q){

int i;

struct PCB \*p = NULL; /\*p为待运行队列PCB指针\*/

head = tail = NULL; /\*初始化头尾指针\*/

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

p = (struct PCB\*)malloc(sizeof(struct PCB)); //分配空间，让p指向这个PCB

init(p,i); //初始化进程

p->next = NULL;

if(head == NULL){ //连接进程

tail = head = p;

p->pre = NULL;

}else{

p->pre = tail;

tail->next = p;

tail = p;

}

}

return p;

}

//冒泡排序链表

void sort(struct PCB \*phead){

struct PCB \*a, \*b;

int i;

for(i=0;i<num;i++){ //外循环循环次数取决于全局变量num，相当于链表长度

a = head; //初始化进程a为head

b = head->next; //初始化进程b为head->next

while(b != NULL){ //b非空

if(a->prior\_num <b->prior\_num){ //比较进程a,b的优先数

exchange(a,b); //调用exchange函数交换a,b进程指针

a = a->pre; //实现一次指针交换要重置a,b进程位置

b = b->next; //因为指针交换数据不变

}

a = a->next; //完成一次内循环后

b = b->next; //指针顺延

}

}

}

//交换相邻两个进程的指针

void exchange(struct PCB \*p, struct PCB \*max){

if(p == max | max != p->next){ //判断若两个进程不相邻，则返回

return;

}

if(max == p->next){ //进程相邻

if(p->pre != NULL){ //进程p指向前一进程的指针不为空

p->pre->next = max; //将p的前一进程的后指针指向max

}else{ //进程p的前一进程指针为空。则说明p原是head指向的进程

head = max; //将头指针指向max

}

if(max->next != NULL){ //max的下一进程指针不为空

max->next->pre=p; //将max的下一进程的前指针指向p

}

max->pre = p->pre; //将max的前一指针指向p的前一指针

p->next = max->next; //将p的后一指针指向max的后一指针

max->next = p; //max的后一指针指向p

p->pre = max; //p的前一指针指向max

}//实现冒泡排序中进程结构体只转换指针，不转换数据

}

//输出进程信息

void showinfor(struct PCB \*phead){

struct PCB \*p;

for(p = phead; p != NULL; p =p->next){

printf("进程 %s\t 优先数 %d\t 运行时间 %d\t 状态 %c\n",p->name,p->prior\_num,p->run\_time,p->status);

}

}

//运行函数

void run(struct PCB \*p){

//输出这个任务的状态，全局时间加1，任务还需要的时间减1，优先级减1

global\_time++; //全局时间加1

p->run\_time--; //运行时间减1

if(p->prior\_num > 0) // 优先数需大于0

p->prior\_num--; //优先数减1

printf("第 %d 次运行:\n",global\_time);

printf("当前进程: %s\t 优先数 %d\t 运行时间:%d\n", p->name, p->prior\_num, p->run\_time);

printf("\n");

}

//判断运行时间是否为0

void check\_runtime(struct PCB \*p){

if(p->run\_time <= 0){ //当运行时间为0，结束进程

p->status = 'E'; //修改进程状态

printf("进程 %s 已结束",p->name);

printf("\n");

printf("进程当前状态为：\n");

showinfor(head); //显示进程信息

printf("\n");

printf("-----------------------------------------------------\n");

printf("请按回车键进行下一进程");

printf("\n");

getchar(); //接收回车键

free(p); //释放p内存

head = p->next; //将头指针顺延

}

}

int main(){

struct PCB \*p = NULL; /\*p为待运行队列PCB指针\*/

//初始化进程链表

PCBinit(p);

//按优先数递减进行进程链表排序

sort(head);

printf("进程当前状态为：\n");

showinfor(head); //显示进程信息

printf("\n");

//若链表中还有PCB是执行循环语句

while(head != NULL){

p = head; //p指向第一个进程

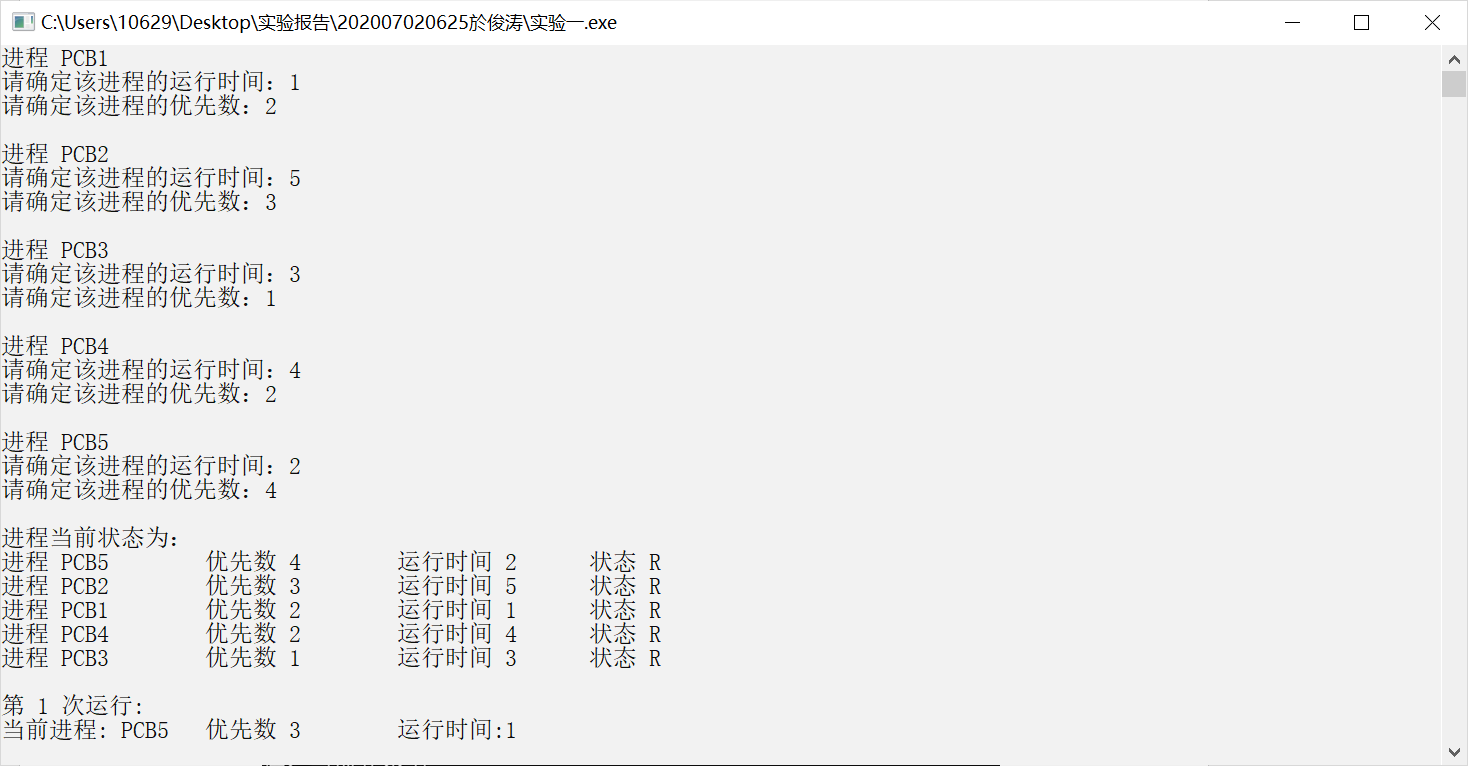
run(p); //进行调度

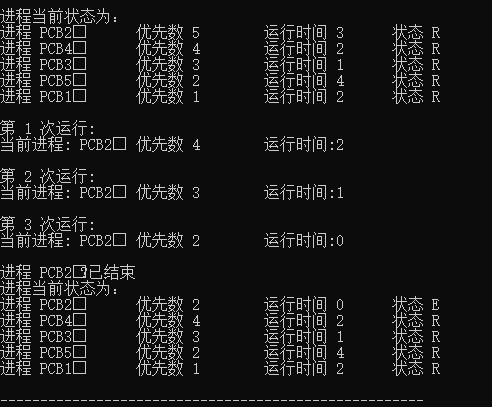
check\_runtime(p); //判断运行时间是否为0

}

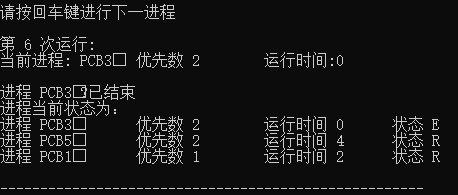
}

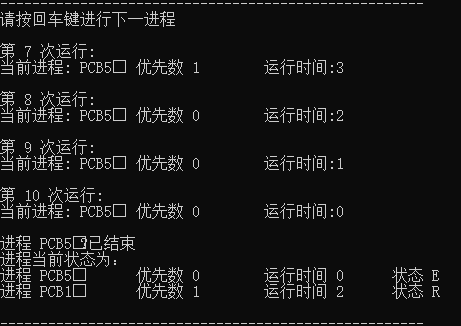
**六、打印的程序运行时初值和运行结果**

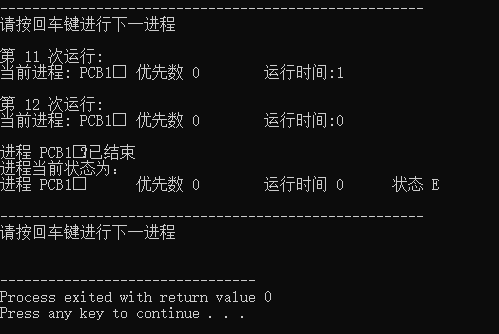












**七、实验小结**

此实验主要是用到链表来实现。用到的数据结构为：

struct PCB

{

char name[10];

int run\_time;

int prior\_num;

char status;

struct PCB \*pre;

struct PCB \*next;

};

先对五个进程按其优先级进行排序，然后找到优先级最高的进程执行。假设当中有进程执行完成，则要从链表中将其删除，并标志出来。不断执行直到进程执行完成。其难点在于对链表的排序及删除操作。