# 广州大学操作系统课程设计实验报告



**课程名称 操作系统课程设计**

**班级 网络173**

**学号 1708200022**

**姓名 庄锦华**

**指导老师 陈康民、张汛涞**

**设计时间 2019/12/26**

**广州大学学生实验报告**

**开课学院及实验室：**计算机科学与工程实验室416A **2019年12月26日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院** | 计算机科学与网络工程学院 | **年级/专业/班** | 网络173 | **姓名** | 庄锦华 | **学号** | 1708200022 |
| **实验课程名称** | 操作系统课程设计 | | | | | **成绩** |  |
| **实验项目名称** | 时间片轮转法 | | | | | **指导老师** | 陈康民、张汛涞 |

1. **实验目的**
2. 学习掌握操作系统的时间片轮转法，实现处理机调度的程序。
3. 掌握进程的状态，包括就绪，运行以及堵塞状态。
4. 学习掌握PCB进程控制块的格式，以及各个参数的意义。
5. **实验内容**
6. 假设系统有n个进程，每个进程用一个进程控制块（PCB）来代表。进程控制块的格式如下表所示，且参数意义也相同。

|  |
| --- |
| 进程名 |
| 链接指针 |
| 到达时间 |
| 估计运行时间 |
| 进程状态 |

1. 按照进程到达的先后顺序排成一个循环队列，设一个队首指针指向第一个到达进程的首址。另外再设一个当前运行进程指针，指向当前正运行的进程。
2. 执行处理机调度时，首先选择队首的第一个进程运行。
3. 由于本题目是模拟实验，所以对被选中的进程并不实际启动运行，而只是执行如下操作：1）估计运行时间减1；

2）输出当前运行进程的名字。

用这两个操作来模拟进程的一次运行。

1. 进程运行一次后，以后的调度则将当前指针依次下移一个位置，指向下一个进程，即调整当前运行指针指向该进程的链接指针所指进程，以指示应运行进程，同时还应判断该进程的剩余运行时间是否为0，若不为0，则等待下一轮的运行，若该进程的剩余运行时间为0，则将该进程的状态置为完成状态“C”，并退出循环队列。
2. 若就绪队列不为空，则重复上述的步骤（4）和（5）直到所有进程都运行完为止。
3. 在所设计的调度程序中，应包含显示或打印语句，以便显示或打印每次选中进程的名称及运行一次后队列的变化情况。
4. **实验思路**
5. 使用PCB结构体存放进程控制块的内容；使用CycleQueue结构体用于管理循环队列，循环队列中有一个指向当前循环队列中正在运行的指针，以及当前指针的前一个元素指针，前一个元素指针的作用在移除元素时起作用。
6. 辅助函数有printProcess（PCB process），用于显示process进程；initProcess（PCB processes[ ]）用于初始化所有进程；arriveTimeSub（PCB processes[ ]）用于将当前循环队列中的所有进程的到达时间-1；round\_Robin（CycleQueue& cycleQueue）为时间片轮转法，将循环队列中指向的当前进程的运行时间-1，并返回当前进程是否运行结束。
7. 为了运行效率更高，使用平均时间复杂度为nlog(n)的快速排序，对初始化后的进程数组按照到达时间的先后顺序进行快速排序，因此仅需要从排好序的进程数组中依次取出到达时间为0的进程放到循环队列中即可。
8. 循环队列的插入元素操作：
   1. 当循环队列为空时，也就是循环队列的当前元素指针为NULL，此时直接将循环队列中，将循环队列的当前指针和当前指针的前一指针指向该元素。
   2. 当循环队列仅有一个元素时，也就是循环队列中当前指针和前一指针指向同一个元素，此时将插入元素设置为循环队列中的前一指针，该插入元素指向的后一个元素为当前指针所指向的元素。
   3. 当循环队列中有多个元素时，
9. 循环队列的删除元素操作：
   1. 当循环队列为空时，操作错误，输出“错误，队列为空。
   2. 当循环队列仅有一个元素时，将循环队列中的当前指针和前一指针设置为NULL即可。
   3. 当循环队列中有两个元素时，删除当前指针元素，将循环队列中的当前指针指向前一元素，循环队列中的两个指针均指向同一个元素。
   4. 当循环队列中有大于两个元素时，将前一指针指向当前元素的下一个元素，完成删除当前指针指向的元素的操作。
10. 每次运行时，首先判断是否有到达时间小于等于0的进程，如果有，则将进程插入到循环队列中，如果当前循环队列非空，则执行一次时间片。判断执行完时间片之后的进程是否已完成，若完成则将当前进程移出循环队列中。

**四、实验过程**

程序中使用的数据结构及主要符号说明

宏定义：

#define NAME 20 //进程名长度

#define n 3 //系统进程数目

#define ARRIVETIME 5 //最长到达时间

#define RUNTIME 5 //最长运行时间

#define ROUND\_ROBIN\_TIME 1//时间片的大小

结构体：

struct PCB {

char name[NAME]; //进程名

PCB\* next = NULL; //链接指针

int arriveTime; //到达时间

int runTime; //估计运行时间

char status; //进程状态

};

struct CycleQueue {

PCB\* preProcess = NULL; //当前进程的前一个进程

PCB\* present = NULL; //当前进程

};

程序流程图

源程序

#include<stdlib.h>

#include<stdio.h>

#include<time.h>

#include<string.h>

#define NAME 20//进程名长度

#define n 3//系统进程数目

#define ARRIVETIME 5//最长到达时间

#define RUNTIME 5//最长运行时间

#define ROUND\_ROBIN\_TIME 1//时间片的大小

/\*

记录错误：

1.函数传递参数，读操作可按值传递；修改操作需要用别名传递

2.插入删除队列元素，队列分一下情况

①为空

②队列仅有一个元素

③队列中有两个元素

④队列中有多个元素

\*/

//进程控制块（PCB）

struct PCB {

char name[NAME];//进程名

PCB\* next = NULL;//链接指针

int arriveTime;//到达时间

int runTime;//估计运行时间

char status;//进程状态

};

//循坏队列

struct CycleQueue {

PCB\* preProcess = NULL;//当前进程的前一个进程

PCB\* present = NULL;//当前进程

};

//显示进程

void printProcess(PCB process) {

printf("进程名为:%s\t\t", process.name);

printf("进程到达的时间为:%d\t\t", process.arriveTime);

printf("进程估计运行时间为:%d\t\t", process.runTime);

printf("进程状态为:%c\n", process.status);

}

//初始化进程

void initProcess(PCB processes[]) {

srand(time(0));//随机种子，用于产生随机到达时间，估计运行时间

char name[NAME] = "PCB";

char num[5];//整型转化为字符串型

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

\_itoa\_s(i+1, num, 5);//转化函数

strcpy\_s(processes[i].name, name);//复制

strcat\_s(processes[i].name, num);//添加为PCB+数字

processes[i].arriveTime = rand() % ARRIVETIME;

processes[i].runTime = rand() % RUNTIME + 1;

processes[i].status = 'N';

printProcess(processes[i]);

}

}

//快速排序算法

int partition(PCB list[], int low, int high) {

int pivotkey = list[low].arriveTime;//选择需要排序的第一个数作为枢轴值

int low\_ = low;//记录左下标

PCB tmp;//用于交换的中间变量

while (low < high) {

//寻找右边第一个小于枢轴值的下标

while (low < high&&list[high].arriveTime >= pivotkey)

high--;

//寻找第一个大于枢轴值的下标

while (low < high&&list[low].arriveTime <= pivotkey)

low++;

//交换两个数值

tmp = list[low];

list[low] = list[high];

list[high] = tmp;

}

//将枢轴值放到枢轴中心

tmp = list[low\_];

list[low\_] = list[low];

list[low] = tmp;

return low;//返回枢轴中心下标

}

//进行快排

void quickSort(PCB list[], int low, int high) {

if (low < high) {

int position = partition(list, low, high);//得到枢轴中心

quickSort(list, low, position - 1);//以枢轴中心为中点，左侧的数组进行快排

quickSort(list, position + 1, high);//以枢轴中心为中点，右侧的数组进行快排

}

}

//所有进程当前时间减一个时间片

void arriveTimeSub(PCB processes[]) {

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

processes[i].arriveTime -= ROUND\_ROBIN\_TIME;

}

//时间片轮转法,若有进程运行结束，则返回true。若没有进程结束，则返回false

bool round\_Robin(CycleQueue& cycleQueue) {

printf("准备执行:");

printProcess(\*cycleQueue.present);//显示当前进程信息

(cycleQueue.present->runTime) -= ROUND\_ROBIN\_TIME;//估计运行时间减1

if (cycleQueue.present->runTime <= 0) {//进程的剩余运行时间为0,退出循环队列

printf("当前进程%s运行结束,退出循环队列\n", cycleQueue.present->name);

cycleQueue.present->status = 'C';//将该进程的状态置为完成状态“C”

return true;

}

return false;

}

//就绪队列插入一个进程，先插入后移动

//循坏队列为空，则没有元素，若present和preProcess相同，则仅有一个元素

void insertCycleQueue(CycleQueue& cycleQueue,PCB& process) {

//循坏队列为空

if (cycleQueue.present == NULL) {

//当前进程

cycleQueue.present = &process;

cycleQueue.present->next = &process;

//前一个进程

cycleQueue.preProcess = &process;

cycleQueue.preProcess->next = &process;

return;

}

//循坏队列仅有一个,\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (cycleQueue.present == cycleQueue.preProcess) {

PCB& presentProcess = \*cycleQueue.present;

cycleQueue.preProcess = &process;

cycleQueue.preProcess->next = &presentProcess;

cycleQueue.present = &presentProcess;

cycleQueue.present->next = cycleQueue.preProcess;

return;

}

//循坏队列有多个

cycleQueue.preProcess->next = &process;

cycleQueue.preProcess = cycleQueue.preProcess->next;

process.next = cycleQueue.present;

}

//将当前进程移出循坏队列

void removeFromCycleQueue(CycleQueue& cycleQueue) {

if (cycleQueue.present == NULL) {

printf("错误！队列为空\n");

return;

}

//循坏队列仅有一个元素

if (cycleQueue.preProcess == cycleQueue.present) {

cycleQueue.preProcess = NULL;

cycleQueue.present = NULL;

return;

}

//循坏队列仅有两个元素

if (cycleQueue.preProcess == cycleQueue.present->next) {

cycleQueue.preProcess->next = cycleQueue.preProcess;

cycleQueue.present = cycleQueue.preProcess;

return;

}

//循坏队列有多个元素,\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* present发生变化

cycleQueue.preProcess->next = cycleQueue.present->next;

cycleQueue.present->next = NULL;

cycleQueue.present = cycleQueue.preProcess->next;

}

int main() {

PCB processes[n];//进程集合

CycleQueue cycleQueue;//循坏队列

int index = 0;//进程集合下标

bool completed = false;//是否有进程完成

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*初始化进程如下：\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

initProcess(processes);//初始化进程

quickSort(processes, 0, n - 1);//使用快速排序

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*排序之后进程如下：\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

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

printProcess(processes[i]);

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*时间片轮转法：\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

while (index != n || cycleQueue.present != NULL) {

completed = false;

//判断是否有到达时间为0的，加入到循坏队列中

while (processes[index].arriveTime <= 0) {

printf("%s进入了循坏队列\n", processes[index].name);

insertCycleQueue(cycleQueue, processes[index]);//插入到循坏队列中

index++;

}

//循坏队列非空

if (cycleQueue.present != NULL) {

//时间片轮转法

completed = round\_Robin(cycleQueue);

//如果有进程完成

if (completed) {

//当前进程移出循坏队列

removeFromCycleQueue(cycleQueue);

}

else {//如果没有进程完成

cycleQueue.preProcess = cycleQueue.preProcess->next;//指针移动到下一个

cycleQueue.present = cycleQueue.present->next;//指针移动到下一个

}

}

arriveTimeSub(processes);//达到时间减一个时间片

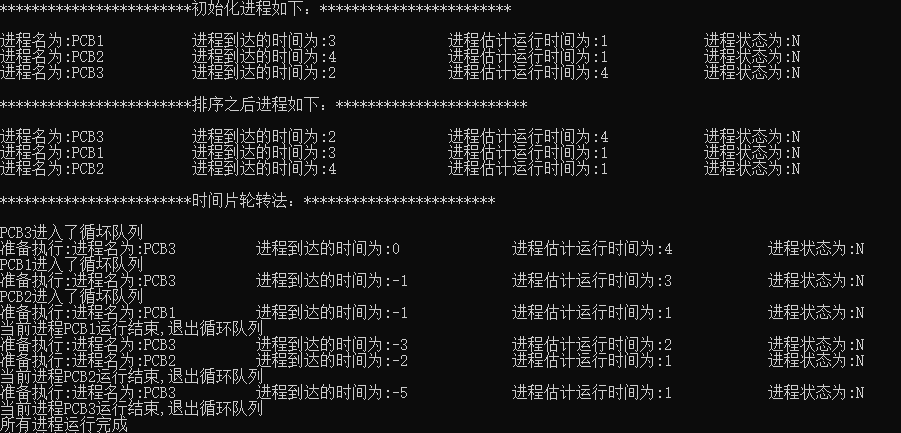
}

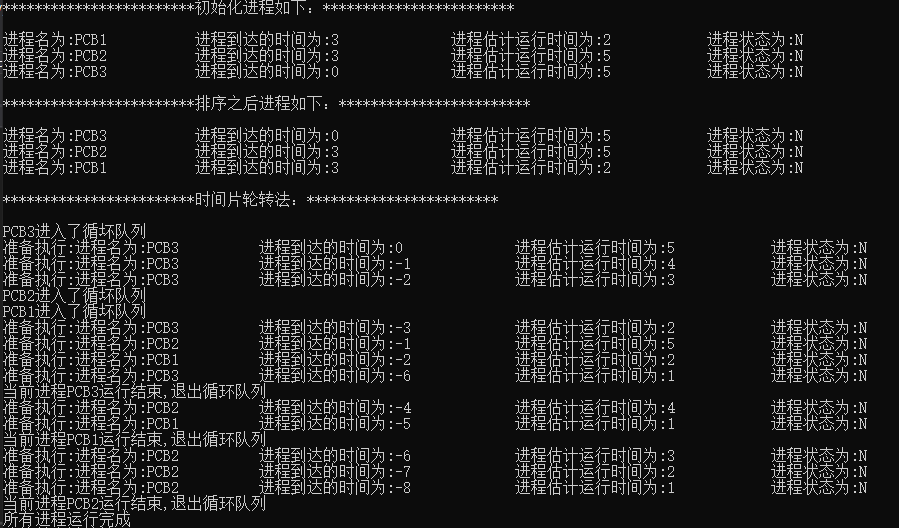
printf("所有进程运行完成\n");

system("pause");

}

1. **实验结果与分析**





1. **实验心得与体会**

实验结果分析

1. 第一次初始化三个进程，进行快速排序之后按时间到达时间顺序为PCB3、PCB1、PCB2，各个进程的关系参数如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 进程名 | 进程到达时间 | 进程估计运行时间 | 进程状态 |
| PCB3 | 2 | 4 | N |
| PCB1 | 3 | 1 | N |
| PCB2 | 4 | 1 | N |

因为PCB3最先到达，因此先运行PCB3，运行完一个时间片之后，PCB3进行就绪队列。此时PCB1进入队列就绪队列，接下来选择PCB3进程执行，执行完成之后，将进程1进入就绪队列，此时进程2进入就绪队列。执行的先后顺序如下：

|  |  |  |
| --- | --- | --- |
| 队列首部 🡨 队列尾部 | | |
| PCB3 |  |  |
| PCB3 | PCB1 |  |
| PCB1 | PCB3 | PCB2 |
| PCB3 | PCB2 |  |
| PCB2 | PCB3 |  |
| PCB3 |  |  |

1. 第二次初始化三个进程，进行快速排序之后按时间到达时间顺序为PCB3、PCB2、PCB1，各个进程的关系参数如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 进程名 | 进程到达时间 | 进程估计运行时间 | 进程状态 |
| PCB3 | 0 | 5 | N |
| PCB2 | 3 | 5 | N |
| PCB1 | 3 | 2 | N |

因为PCB3最先到达，因此先运行PCB3，由于第二个进程的达到时间为3，因此PCB3会先执行三次时间片，然后PCB3进入就绪队列。紧接着PCB2和PCB1依次进入就绪队列，执行的先后顺序如下：

|  |  |  |
| --- | --- | --- |
| 队列首部 🡨 队列尾部 | | |
| PCB3 |  |  |
| PCB3 |  |  |
| PCB3 |  |  |
| PCB3 | PCB2 | PCB1 |
| PCB2 | PCB1 | PCB3 |
| PCB1 | PCB3 | PCB2 |
| PCB3 | PCB2 | PCB1 |
| PCB2 | PCB1 |  |
| PCB1 | PCB2 |  |
| PCB2 |  |  |
| PCB2 |  |  |
| PCB2 |  |  |

实验收获和体会

1. 在做操作系统课程设计的时间片轮转法时，进一步了解了PCB进程控制块以及进程控制块的各个参数意义，使用了循环队列实现处理机调度的程序，进一步熟悉了队列的使用。
2. 在进行排序时，使用了平均时间复杂度为nlog（n）的快速排序算法，使得效率查找进程进入队列的效率更加好，减少了每次遍历循环查找是否需要有进程加入就绪队列的操作，进一步熟悉了快速排序的使用。
3. 本次实现的过程中，遇到了插入删除操作出现bug的问题，在进行插入删除时，应分为循环队列中为空、仅有一个元素等情况。让自己对此类问题有了更深刻的印象，处理类似操作时应该分多种情况处理。
4. 进行参数传递时，应该注意区别按值传递、按别名传递。对读取操作，可以使用按值传递，对于修改操作，应该使用别名传递。

实验的改进意见和建议

1. 在进程完成运行任务时，将进程移出队列。因此可以考虑使用new方式创建进程，这样可以在进程移出队列时使用delete将创建的进程的空间回收掉，减少栈变量存储的空间。
2. 由于是模拟时间片轮转法，因此所有的进程在一开始便随机产生了，可以使用快速排序的方法进行排序，在排序好的进程中依次选取入队。但是现实中进程的创建的时间是不确定的，在中途可能会有新创建的进程加入，因此这种情况下不可使用排序算法进行排序。