

**学生实验报告**

实验课名称：操作系统

实验项目名称：进程调度

专业名称：计算机科学与技术（交通信息工程）

班级：2016240203

学号：2016902094

学生姓名：瞿强鑫

教师姓名：刘晓春

2019**年** 5 **月** 21**日**

1. 实验名称

进程调度

1. 实验内容

要求：学生通过实验加深理解进程调度的特点，并掌握几种常见的调度算法。

内容：编写模拟程序实现FIFO算法，短作业优先调度算法和轮转法，比较他们的不同之处。

1. 实验原理和设计思路

不同的调度算法，平均周转时间和平均带权周转时间有很大差别，应根据具体情况具体选择。

FIFO（First In First Out）先进先出算法是一种简单的调度算法，既可用于作业调度也可用于进程调度，每次调度都选择最先进入就绪队列的进程调度。

SF（Shortest First）短作业优先调度算法就是估计运行时间最短的作业或本次CPU周期最短的进程执行。

RR（Round Robin）轮转法也称为时间片轮转法，它依照公平对待的原则，照顾到所有的进程，让他们都有机会得到执行，能够保证就绪队列的所有进程在一给定的时间内，均能获得一个时间片的处理机执行时间。

1. 源代码

#include<stdio.h>

#include<string.h>

#include<iostream>

using namespace std;

const int block\_time = 10; //定义时间片的长度为10秒

const int MAXPCB = 100; //定义最大进程数

//定义进程结构体，进程名 所需时间 到达时间 进程完成时间 进程等待时间

typedef struct node

{

char name[20];

int time;

int arr\_time;

int finished;

int wait\_time;

} pcb;

pcb pcbs[MAXPCB];

int quantity;

//初始化函数

void initial()

{

int i;

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

{

strcpy(pcbs[i].name, "");

pcbs[i].time = 0;

pcbs[i].arr\_time = 0;

pcbs[i].finished = 0;

pcbs[i].wait\_time = 0;

}

quantity = 0;

}

//读数据函数

int readData()

{

FILE \*fp;

char fname[20];

int i;

cout << "请输入读取文件名:";

cin >> fname;

if ((fp = fopen(fname, "r")) == NULL)

{

cout << "错误,文件打不开,请检查文件名" << endl;

}

else

{

while (!feof(fp))

{

fscanf(fp, "%s %d %d", pcbs[quantity].name, &pcbs[quantity].time, &pcbs[quantity].arr\_time);

quantity++;

}

//输出所读入的数据

cout << "输出所读入的数据" << endl<<endl<<endl;

cout << "进程名 到达时间 所需时间" << endl;

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

cout << " " << pcbs[i].name << " " << pcbs[i].arr\_time << " " << pcbs[i].time << endl;

}

return (1);

}

return (0);

}

//重置数据,以供另一个算法使用

void init()

{

int i;

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

{

pcbs[i].finished = 0;

pcbs[i].wait\_time = 0;

}

}

//先进先出算法

void FIFO()

{

int i, j;

int total;

//输出FIFO算法执行流

cout << endl << "---------------------------------------------------------------" << endl;

cout << "FIFO算法执行流:" << endl;

cout << "进程名 等待时间" << endl;

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

{

cout << " " << pcbs[i].name << " " << pcbs[i].wait\_time << endl;

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

{

pcbs[j].wait\_time += pcbs[i].time;

}

}

total = 0;

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

{

total += pcbs[i].wait\_time;

}

cout << "总等待时间:" << total << " 平均等待时间:" << total / quantity << endl;

}

//短作业优先算法

void privilege()

{

cout << endl << "---------------------------------------------------------------" << endl;

cout << "短作业优先算法执行流:" << endl;

cout << "进程名 等待时间" << endl;

pcb sort[MAXPCB];

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

{

strcpy(sort[i].name, "");

sort[i].time = 0;

sort[i].arr\_time = 0;

sort[i].finished = 0;

sort[i].wait\_time = 0;

}

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

{

char \*k = pcbs[i+1].name;

strcpy(sort[i].name, k);

sort[i].time = pcbs[i+1].time;

}

for (int i = 1; i <quantity-1; i++)

{

int j, v;

char \*k;

for (v = sort[i].time,k=pcbs[i+1].name, j = i - 1; j >= 0 && v < sort[j].time; j--)

{

sort[j + 1].time = sort[j].time;

strcpy(sort[j + 1].name, sort[j].name);

}

sort[j + 1].time = v;

strcpy(sort[j + 1].name, k);

}

cout << " " << pcbs[0].name << " " << pcbs[0].wait\_time << endl;

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

{

cout << " " << sort[i].name << " " << sort[i].wait\_time << endl;

for (int j = i + 1; j<quantity; j++)

{

sort[j].wait\_time += sort[i].time;

}

}

int total = 0;

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

{

total += sort[i].wait\_time;

}

cout << "总等待时间:" << total << " 平均等待时间:" << total / quantity << endl;

}

//时间片轮转调度算法

void timer()

{

int i, j, number, flag = 1;

int passed\_time = 0;

int max\_time = 0;

int round = 0;

int queue[1000];

int total = 0;

while (flag == 1)

{

flag = 0;

number = 0;

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

{

if (pcbs[i].finished == 0)

{

number++;

j = i;

}

}

if (number>0)

{

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

{

if (pcbs[i].finished == 0)

{

flag = 1;

queue[total] = i;

total++;

if (pcbs[i].time <= block\_time\* (round + 1))

{

pcbs[i].finished = 1;

}

}

}

}

round++;

}

cout << endl << "---------------------------------------------------------------" << endl;

cout << "时间片轮转调度作业顺序：:";

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

{

cout << pcbs[queue[i]].name << " ";

}

cout << endl;

}

//主函数

int main()

{

while (true)

{

cout << endl << endl;

cout << " \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n" << endl;

cout << " 　　　　　　 进程调度模拟 　　　　　　　 \n" << endl;

cout << " \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << endl << endl;

initial();

int flag;

flag = readData();

if (flag == 1)

{

init();

FIFO();

init();

privilege();

init();

timer();

}

cout << endl << endl;

cout << "结束请按0，按任意键继续\n";

char q;

cin >> q;

if (q == '0')

{

break;

}

}

return 0;

}

1. 结果截图

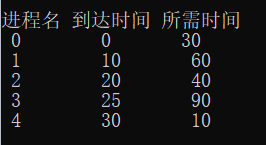


图1.进程模拟

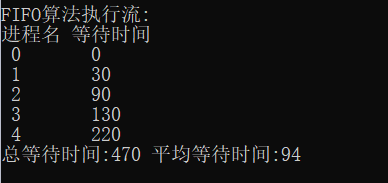


图2. FIFO算法

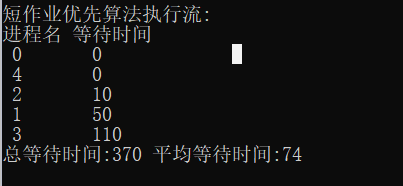


图3. SF短作业优先算法



图4. RR时间片轮转算法

1. 编程体会

本次实验做的是进程调度的模拟实验，在Visual Studio C语言环境下编写，使用的数据结构就是结构体了，针对每个进程的对应属性进程名、到达时间、执行所需时间分别设结构体的三个对应属性用于存储相应的值。FIFO可以说是很简单的，最先到达的进程根据其到达时间属性判断属性值最大即最早到达的进程；SF需要在结构体数组的基础上，判断当前执行的进程过程中到达的进程并排序出这一部分进程中所需时间最小的进程最先执行，在第二个进程执行时判断这个过程到达的进程（包括之前到达未执行的进程），以此类推完成；RR算法则是将时间公平的分给到达的进程们，每个进程轮着使用CPU一个时间片的时间并循环，执行完的进程退出循环，未完成的继续直到完成。

实验虽然算法很简单但实现起来却很难，数据结构必须合适才能最高效的实现!