一、实验目的

实现进程调度算法：

1.先来先服务算法

2.时间片轮转算法

3.优先级队列算法

二、实验原理

1.先来先服务算法：

按照进程的到达时间进行服务，只有当上一个进程运行结束后才会运行下一个进程。记录进程运行的开始执行时间，完成时间，周转时间，带权周转时间。

2.时间片轮转算法：

按照进程到达时间进行服务，每个进程按照时间片进行轮转，记录进程运行的开始执行时间，完成时间，周转时间，带权周转时间。

3.优先级队列算法：

按照进程的优先级进行服务，但是前提在要求时间内到达，否则运行已经到达的进程中优先权最高的，记录进程运行的开始时间，完成时间，周转时间，带权周转时间。

三、实验步骤、数据记录及处理

**1. 算法流程**

float averageTime;//平均带权周转时间

float numRevolveTime;//总带权周转时间

int numTime;//总执行时间

typedef struct PCB

{

char \_name;

int \_arriveTime;//到达时间

int \_serveTime;//服务时间

int \_beginTime;//开始时间

int \_finishTime;//完成时间

int \_revolveTime;//周转时间

float \_valueRevolveTime;//带权周转时间

int \_priority;//优先级

int flage;//标志

struct PCB\* next;

}PCB, \*pPCB;

主程序流程图：



**核心算法：**

以下链表是按照到达时间进行排序

void Running(pPCB\* head)//先来新服务

{

PCB\* pCur = \*head;//pCur指向链表头

if ((\*head) == NULL)

return;

pCur->\_beginTime = pCur->\_arriveTime;//定义第一个进程开始时间为它的到达时间

while (pCur)//遍历链表

{

pCur->\_finishTime = pCur->\_beginTime + pCur->\_serveTime;//计算完成时间

pCur->\_revolveTime = pCur->\_finishTime - pCur->\_arriveTime;//计算周转时间

pCur->\_valueRevolveTime = ((float)pCur->\_revolveTime / pCur->\_serveTime);//进行带权周转时间

numRevolveTime += pCur->\_valueRevolveTime;//计算总带权周转时间

numTime += 1;

if (pCur->next) {//给下一个进程的到达时间进行赋值

pCur->next->\_beginTime = pCur->\_finishTime;

}

pCur = pCur->next;

}

}

void RunningTime(pPCB\* head,int time)//时间片轮转

{

PCB \*pCur = \*head;

PCB \*pT = \*head;

int t = pCur->\_beginTime = pCur->\_arriveTime;//设置第一个开始之间为到达时间

int arr[10];

int n = 0, n1 = 0;

while (pT)//计算进程个数和进程的开始时间

{

arr[n] = pT->\_serveTime;

if (pT->next)

{

pT->next->\_beginTime = time + pT->\_beginTime;

}

pT = pT->next;

n++;

}

while (n)//遍历

{

int i = 0;

pT = pCur;

while (pT )

{

if (pT->\_serveTime > 0 && pT->\_arriveTime <= t)//进程的没有运行结束，并且到达时间在t之前

{

i++;

pT->\_serveTime = pT->\_serveTime - time;//服务时间-时间片

t += time;//总时间增加

if (pT->\_serveTime <= 0)//如果服务时间<0，更改总时间

{

t = t + pT->\_serveTime;

pT->\_finishTime = t;//确定完成时间

n--;

}

}

pT = pT->next;

}

if (i == 0) {//如果没有进程在t之前到达，

pT = pCur;

while (pT)

{

if (pT->\_serveTime >= 0)//进程没有运行结束

{

t = pT->\_arriveTime;

pT->\_beginTime = t;//更改到达时间

break;

}

pT = pT->next;

}

}

}

while (pCur)//遍历

{

pCur->\_serveTime = arr[n1++];//计算服务时间

pCur->\_revolveTime = pCur->\_finishTime - pCur->\_arriveTime;//计算周转时间

pCur->\_valueRevolveTime = (float)pCur->\_revolveTime / pCur->\_serveTime;//计算带权周转时间

numRevolveTime += pCur->\_valueRevolveTime;//总带权周转时间

numTime += 1;

pCur = pCur->next;

}

}

void RunPriority(PCB\* head) {//优先权

PCB\* pCur = head;

PCB\* newHead = NULL;

int num = 0;

int t = pCur->\_arriveTime;

int minPri;

while (pCur)//计算进程个数

{

num++;

pCur = pCur->next;

}

while (num--)//遍历

{

pCur = head;

int n = 0;

while (pCur && pCur->\_arriveTime <= t)//查找到达时间在t之前的进程

{

if (n == 0 && pCur->flage == 0)//第一次进入，判断是否已经查找过过

{

minPri = pCur->\_priority;//记录最小优先级

newHead = pCur;

n++;

}

if (pCur->\_priority < minPri && pCur->flage == 0)//判断优先级是否比最小优先级还要高，并且是否被查找过

{

newHead = pCur;

minPri = newHead->\_priority;

}

pCur = pCur->next;

}//找到优先级最高并且到达时间在t之内

newHead->\_beginTime = t;//计算开始时间

newHead->\_finishTime = newHead->\_beginTime + newHead->\_serveTime;//计算完成时间

newHead->\_revolveTime = newHead->\_finishTime - newHead->\_arriveTime;//计算周转时间

newHead->\_valueRevolveTime = ((float)newHead->\_revolveTime / newHead->\_serveTime);//计算带权周转时间

newHead->flage = 1;//更改标记

numRevolveTime += newHead->\_valueRevolveTime;//计算总带权周转时间

numTime++;

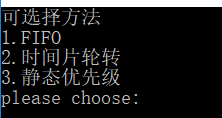
t = newHead->\_finishTime;

}

}

**2. 运行结果分析**

界面：



输入进程：

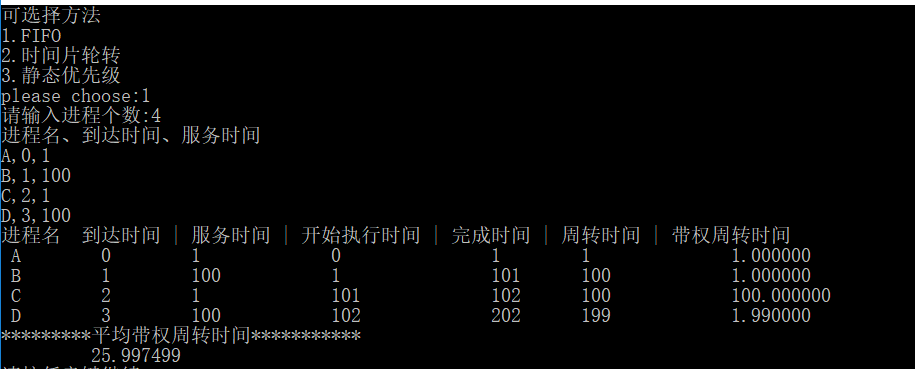
A,0,1

B,1,100

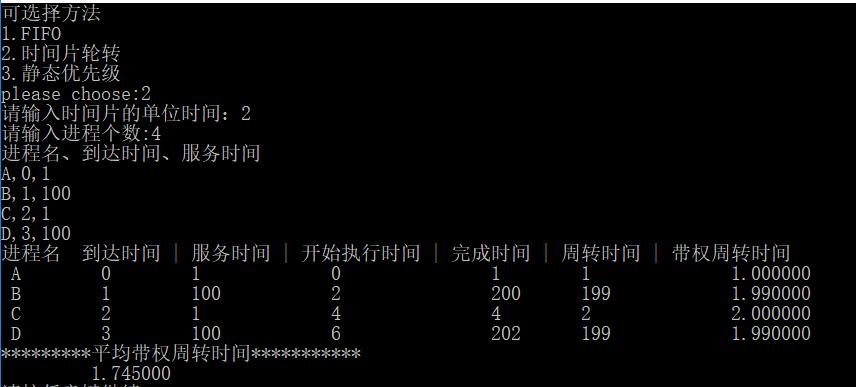
C,2,1

D,3,100

**1.FIFO**



**2.时间片轮转**



**3.静态优先级**

输入进程：

A,0,4

B,2,3

C,3,5

D,5,2

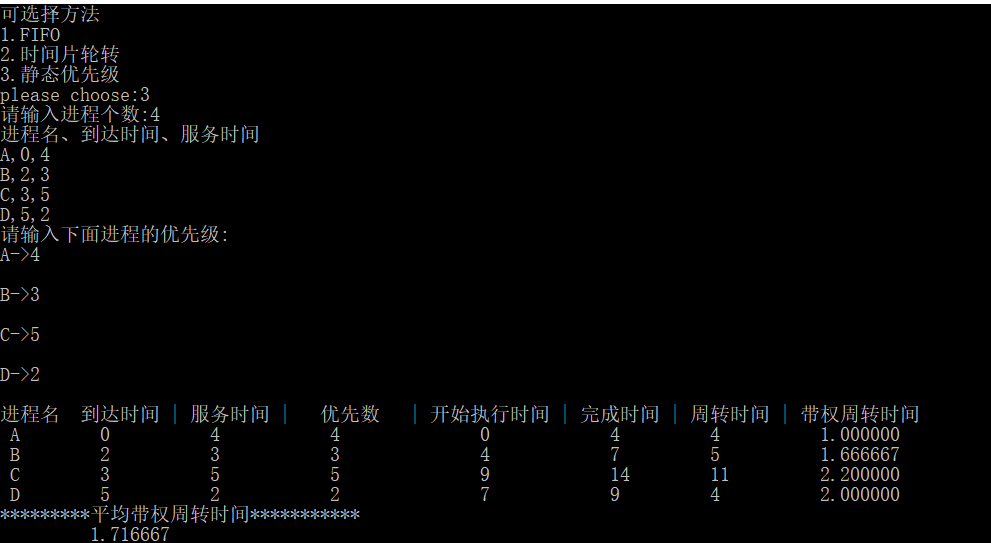
优先权

A->4

B->3

C->5

D->2



四、总结与体会

通过本次实验自己编写程序运行进程调度算法对先来先服务、时间片轮转、优先权调度有了更好的理解，在编程过程中会存在不同算法调用不同函数的情况，所以会出现代码的冗余度较高，之后将多个函数进行合并降低冗余度。在编写过程中scanf\_s的输入存在不能四个进程直接输入，上网查阅后解决了这个问题，最后改为scanf输入，解决缓冲区问题就可以了。

通过本次实验对操作系统的进程调度算法有了更加清晰的理解，加深了对算法的印象，提高了自己的程序编写能力。

# 附录：源代码

**Test.c:**

#include"Operation System.h"

void menu()

{

printf("可选择方法\n");

printf("1.FIFO\n");

printf("2.时间片轮转\n");

printf("3.静态优先级\n");

printf("please choose:");

}

void test()

{

int c;

int time;

menu();

scanf\_s("%d", &c);

switch (c)

{

case 1:

FIFOServe();

break;

case 2:

printf("请输入时间片的单位时间：");

scanf("%d", &time);

TimeRevolve(time);

break;

case 3:

Priority();

break;

default:

printf("重新选择\n");

test();

break;

}

}

int main()

{

test();

system("pause");

}

**Operation System.h:**

#define \_CRT\_SECURE\_NO\_WARNINGS 1

#include<stdio.h>

#include<assert.h>

#include<malloc.h>

#include<stdlib.h>

float averageTime;

float numRevolveTime;

int numTime;

typedef struct PCB

{

char \_name;

int \_arriveTime;

int \_serveTime;

int \_beginTime;

int \_finishTime;

int \_revolveTime;

float \_valueRevolveTime;

int \_priority;

int flage;

struct PCB\* next;

}PCB, \*pPCB;

void print(PCB\* head);

void InitList(pPCB\* head);

void FifoInsertList(pPCB\* head, char n, int arriveTime, int serveTime);

//void FifoDelList(pPCB\* head);

void FIFOServe();

void TimeRevolve(int time);

void RunningTime(pPCB\* head, int time);

**Operation System.c:**

#include"Operation System.h"

pPCB BuyNode(char n, int arriveTime, int serveTime)

{

PCB\* NewNode = (PCB \*)malloc(sizeof(PCB));

if (NewNode == NULL)

{

assert("malloc:BuyNode");

return NULL;

}

NewNode->next = NULL;

NewNode->\_name = n;

NewNode->\_arriveTime = arriveTime;

NewNode->\_serveTime = serveTime;

return NewNode;

}

void InitList(pPCB\* head)

{

assert(head);

(\*head) = NULL;

}

void FifoInsertList(pPCB\* head, char n, int arriveTime, int serveTime)

{

PCB\* pCur = (\*head);

PCB\* pNew = NULL;

PCB\* pPre = NULL;

if ((\*head) == NULL)

{

(\*head) = BuyNode(n, arriveTime, serveTime);

}

else

{

if (pCur->\_arriveTime > arriveTime)

{

pNew = BuyNode(n, arriveTime, serveTime);

pNew->next = pCur;

(\*head) = pNew;

return;

}

while (pCur != NULL && arriveTime >= pCur->\_arriveTime)

{

pPre = pCur;

pCur = pCur->next;

}

pNew = BuyNode(n, arriveTime, serveTime);

pNew->next = pCur;

pPre->next = pNew;

}

}

void CreatList(pPCB\* head)

{

int num, i;

char n;

int arriveTime;

int serveTime;

InitList(head);

printf("请输入进程个数:");

scanf\_s("%d", &num);

printf("进程名、到达时间、服务时间\n");

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

{

scanf(" %c,%d,%d",&n, &arriveTime, &serveTime);

FifoInsertList(head, n, arriveTime, serveTime);

}

}

void Running(pPCB\* head)

{

PCB\* pCur = \*head;

if ((\*head) == NULL)

return;

pCur->\_beginTime = pCur->\_arriveTime;

while (pCur)

{

pCur->\_finishTime = pCur->\_beginTime + pCur->\_serveTime;

pCur->\_revolveTime = pCur->\_finishTime - pCur->\_arriveTime;

pCur->\_valueRevolveTime = ((float)pCur->\_revolveTime / pCur->\_serveTime);

numRevolveTime += pCur->\_valueRevolveTime;

numTime += 1;

if (pCur->next) {

pCur->next->\_beginTime = pCur->\_finishTime;

}

pCur = pCur->next;

}

}

void print(PCB\* head)

{

int n = 0;

printf("进程名\t到达时间 | 服务时间 | 开始执行时间 | 完成时间 | 周转时间 | 带权周转时间\n");

while (head != NULL)

{

printf(" %c\t %d\t %d\t\t %d\t\t %d\t %d\t\t %f\t\n",

head->\_name, head->\_arriveTime, head->\_serveTime, head->\_beginTime, head->\_finishTime, head->\_revolveTime, head->\_valueRevolveTime);

head = head->next;

n++;

}

averageTime = (numRevolveTime / n);

printf("\*\*\*\*\*\*\*\*\*平均带权周转时间\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t %f\n", averageTime);

}

void FIFOServe()

{

PCB\* head;

CreatList(&head);

Running(&head);

print(head);

}

void RunningTime(pPCB\* head,int time)

{

PCB \*pCur = \*head;

PCB \*pT = \*head;

int t = pCur->\_beginTime = pCur->\_arriveTime;

int arr[10];

int n = 0, n1 = 0;

while (pT)

{

arr[n] = pT->\_serveTime;

if (pT->next)

{

pT->next->\_beginTime = time + pT->\_beginTime;

}

pT = pT->next;

n++;

}

while (n)

{

int i = 0;

pT = pCur;

while (pT )

{

if (pT->\_serveTime > 0 && pT->\_arriveTime <= t)

{

i++;

pT->\_serveTime = pT->\_serveTime - time;

t += time;

if (pT->\_serveTime <= 0)

{

t = t + pT->\_serveTime;

pT->\_finishTime = t;

n--;

}

}

pT = pT->next;

}

if (i == 0) {

pT = pCur;

while (pT)

{

if (pT->\_serveTime >= 0)

{

t = pT->\_arriveTime;

pT->\_beginTime = t;

break;

}

pT = pT->next;

}

}

}

while (pCur)

{

pCur->\_serveTime = arr[n1++];

pCur->\_revolveTime = pCur->\_finishTime - pCur->\_arriveTime;

pCur->\_valueRevolveTime = (float)pCur->\_revolveTime / pCur->\_serveTime;

numRevolveTime += pCur->\_valueRevolveTime;

numTime += 1;

pCur = pCur->next;

}

}

void TimeRevolve(int time)

{

PCB\* head;

CreatList(&head);

RunningTime(&head,time);

print(head);

}

void setPriority(PCB\* head) {

int p = 0;

printf("请输入下面进程的优先级:\n");

PCB\* pCur = head;

while (pCur) {

printf("%c->", pCur->\_name);

scanf("%d", &pCur->\_priority);

printf("\n");

pCur->flage = 0;

pCur = pCur->next;

}

}

void RunPriority(PCB\* head) {

PCB\* pCur = head;

PCB\* newHead = NULL;

int num = 0;

int t = pCur->\_arriveTime;

int minPri;

while (pCur)

{

num++;

pCur = pCur->next;

}

while (num--)

{

pCur = head;

int n = 0;

while (pCur && pCur->\_arriveTime <= t)

{

if (n == 0 && pCur->flage == 0)

{

minPri = pCur->\_priority;

newHead = pCur;

n++;

}

if (pCur->\_priority < minPri && pCur->flage == 0)

{

newHead = pCur;

minPri = newHead->\_priority;

}

pCur = pCur->next;

}

newHead->\_beginTime = t;

newHead->\_finishTime = newHead->\_beginTime + newHead->\_serveTime;

newHead->\_revolveTime = newHead->\_finishTime - newHead->\_arriveTime;

newHead->\_valueRevolveTime = ((float)newHead->\_revolveTime / newHead->\_serveTime);

newHead->flage = 1;

numRevolveTime += newHead->\_valueRevolveTime;

numTime++;

t = newHead->\_finishTime;

}

}

void Priority() {

PCB\* head;

int n = 0;

CreatList(&head);

setPriority(head);

RunPriority(head);

printf("进程名\t到达时间 | 服务时间 | 优先数 | 开始执行时间 | 完成时间 | 周转时间 | 带权周转时间\n");

while (head != NULL)

{

printf(" %c\t %d\t %d\t\t %d\t\t%d\t %d\t %d\t %f\t\n",

head->\_name, head->\_arriveTime, head->\_serveTime, head->\_priority,head->\_beginTime, head->\_finishTime, head->\_revolveTime, head->\_valueRevolveTime);

head = head->next;

n++;

}

averageTime = (numRevolveTime / n);

printf("\*\*\*\*\*\*\*\*\*平均带权周转时间\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t %f\n", averageTime);