**实验五 进程调度**

# 姓 名 陈正江 学 号 19001531 成绩

实验时间 2023.11.23 指导教师(签名)

**（诚信声明：本实验报告内容，均由本人亲自上机完成。 签名：\_\_\_\_\_\_\_\_\_\_\_\_\_）**

一．实验目的

1.深入了解进程调度的策略和机制

2.掌握各种调度算法的基本原理和评价指标

3.实现几种常用的调度算法

二．实验工具与设备

装有 Linux 操作系统的计算机。

三．实验内容

1、用 C 语言实现 FCFS、SPN 和 RR 算法

**（1）程序框架设计**

#include<stdio.h>

#include<stdlib.h>

**// 先来先服务、短进程优先算法的结构体**

**typedef struct process\_FCFS\_SPN**{

char name; // 进程名

float arrivetime; // 到达时间

float servetime; // 服务时间

float finishtime; // 完成时间

float roundtime; // 周转时间

float daiquantime; // 带权周转时间

struct process\_FCFS \_SPN \*link; // 结构体指针

}FCFS\_SPN;

**// 时间片轮转算法的结构体**

**typedef struct RR**{

char name; // 进程名

float arrive; // 进程到达时间

float run; // 进程运行时间

float rest; // 进程剩余时间

char \*state; // 进程状态

struct RR \*next; // 结构体指针

}RR;

FCFS\_SPN \*p, \*q, \*head = NULL;

struct RR\*front = NULL, \*rear = NULL;

struct process\_FCFS\_SPN a[100];

float avrRoundtime; // 平均周转时间

float avrWeighted\_time; // 平均带权周转时间

**// 按到达时间从小到大进行冒泡排序**

**struct process\_FCFS\_SPN \*sortarrivetime(struct process\_FCFS\_SPN a[], int n)**{

int i, j;

struct process\_FCFS\_SPN temp;

int flag;

for(i = 1; i < n; i++){

flag = 0;

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

if(a[j+1].arrivetime < a[j].arrivetime) { // 把到达时间小的换到前面

temp = a[j]; a[j] = a[j+1]; a[j+1] = temp;

flag = 1; // 交换成功

}

}if(flag == 0) break; // 如果一次排序中没有发生任何交换，则排序结束

}return a; // 返回排好序的进程数组

}

**// 按服务时间从小到大进行冒泡排序**

**struct process\_FCFS\_SPN \*sortservetime(struct process\_FCFS\_SPN a[], int n)**{

int i = 0, j = 0;

struct process\_FCFS\_SPN temp;

int flag;

for(i = 1; i < n; i++){

flag = 0;

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

if(a[j+1].servetime < a[j].servetime) { //将服务时间短的换到前面

temp = a[j]; a[j] = a[j+1]; a[j+1] = temp;

flag = 1; // 交换成功

}

}if(flag == 0) break; // 如果一次排序中没有发生任何交换，则排序结束

}return a; // 返回排好序的进程数组

}

**// 创建时间片轮转调度队列**

**struct RR \*create(int &a)**{

int i;

struct RR \*head, \*rear, \*p, \*q, \*t;

head = rear = NULL;

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

p = (struct RR\*)malloc(sizeof(struct RR));

printf("%d 进程名：", i+1); scanf("%s", &p->name);

printf("到达时间："); scanf("%f", &p->arrive);

printf("服务时间："); scanf("%f", &p->run);

p->rest = p->run;

p->state = (char\*)"ready";

if(rear == NULL) {

head = p; p->next = NULL; rear = p;

}

else{

t = NULL;

q = head;

while(q && q->arrive < p->arrive) {

t = q; q = q->next;

}

if(q == head) {

p->next = head; head = p;

}

else if(t == rear) {

rear->next = p; p->next = NULL; rear = p;

}

else { // 将P放在中间

t->next = p; p->next = q;

}

}

}return head;

}

**void InsertQue(struct RR \*temp)**{  **// 调度队列的进程插入**

if(rear != NULL){

rear->next = temp; rear = rear->next;

}

else{

rear = temp; front = rear;

}

}

**struct RR \*DelQue()**{ **// 调度队列的单个进程的删除**

struct RR \*temp = NULL;

if(front == NULL)

return NULL;

else{

temp = front;

front = front->next;

if(front == NULL)

rear = NULL;

}return temp;

}

**（2）先来先服务调度算法**

**void FCFS (struct process\_FCFS\_SPN a[], int n, float &t1, float &t2)**{

int i;

a[0].finishtime = a[0].arrivetime + a[0].servetime; // 完成时间 = 到达时间 + 服务时间

a[0].roundtime = a[0].finishtime - a[0].arrivetime; // 周转时间 = 完成时间 - 到达时间

a[0].weighted\_time = a[0].roundtime / a[0].servetime; // 带权时间 = 周转时间 / 服务时间

for(i = 1; i < n; i++){

if(a[i].arrivetime < a[i-1].finishtime) {// 若当前进程的到达时间在上一个进程结束时间之前

a[i].finishtime = a[i-1].finishtime + a[i].servetime; // 完成时间

a[i].roundtime = a[i].finishtime - a[i].arrivetime; // 周转时间

a[i].weighted\_time = a[i].roundtime / a[i].servetime; // 带权时间

}

else{ // 若当前进程的到达时间在上一个进程结束时间之后

a[i].finishtime = a[i].arrivetime + a[i].servetime; // 完成时间

a[i].roundtime = a[i].finishtime - a[i].arrivetime; // 周转时间

a[i].weighted\_time = a[i].roundtime / a[i].servetime; // 带权时间

}

}

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

printf("进程名：%c ", a[i].name);

printf("到达时间：%.3f ", a[i].arrivetime);

printf("服务时间：%.3f ", a[i].servetime);

printf("完成时间：%.3f ", a[i].finishtime);

printf("周转时间：%.3f ", a[i].roundtime);

printf("带权周转时间：%.3f ", a[i].weighted\_time);

printf("\n");

t1 += a[i].roundtime; t2 += a[i].weighted\_time;

}

}

**（3）短进程优先调度算法**

**void SPN(struct process\_FCFS\_SPN a[], int n, float &t1, float &t2)**{

int i;

a[0].finishtime = a[0].arrivetime + a[0].servetime; // 完成时间 = 到达时间 + 服务时间

a[0].roundtime = a[0].finishtime - a[0].arrivetime; // 周转时间 = 完成时间 - 到达时间

a[0].weighted\_time = a[0].roundtime / a[0].servetime; // 带权时间 = 周转时间 / 服务时间

for(i = 1; i < n; i++) {

if(a[i].arrivetime < a[i-1].finishtime) {// 若当前进程的到达时间在上一个进程结束时间之前

a[i].finishtime = a[i-1].finishtime + a[i].servetime; // 完成时间

a[i].roundtime = a[i].finishtime - a[i].arrivetime; // 周转时间

a[i].weighted\_time = a[i].roundtime / a[i].servetime; // 带权时间

}

else { // 若当前进程的到达时间在上一个进程结束时间之后

a[i].finishtime = a[i].arrivetime + a[i].servetime; // 完成时间

a[i].roundtime = a[i].finishtime - a[i].arrivetime; // 周转时间

a[i].weighted\_time = a[i].roundtime / a[i].servetime; // 带权时间

}

}

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

printf("进程名：%c ", a[i].name);

printf("到达时间：%.3f ", a[i].arrivetime);

printf("服务时间：%.3f ", a[i].servetime);

printf("完成时间：%.3f ", a[i].finishtime);

printf("周转时间：%.3f ", a[i].roundtime);

printf("带权周转时间：%.3f ", a[i].weighted\_time);

printf("\n");

t1 += a[i].roundtime; t2 += a[i].weighted\_time;

}

}

**（4）时间片轮转调度算法 RR**

**void RR(struct RR \*head, int &a)**{

struct RR \*p, \*t, \*r;

float slice = 0.0f;

float temp = 0.0f; // 缓存最后一个正数rest

float m1 = 0.0f;

float sum\_round = 0.0f, sum\_weighted\_ = 0.0f; // 所有进程总周转时间，所有进程总带权周转时间

float round = 0.0f, weighted\_ = 0.0f; // 周转时间，带权周转时间

float avr\_round = 0.0f, avr\_weighted\_ = 0.0f;

printf("请输入时间块大小: ");

scanf("%f", &slice);

t = head;

head = head->next;

t->next = NULL;

InsertQue(t); // 进程队列中的第一个进程先放入队列里面

while(front != NULL) { // 不断循环，直到队列为空

p = DelQue(); // 从队列中取出第一个进程

m1 += slice;

temp = p->rest;

p->rest = p->rest - slice; // 剩余时间

p->state = (char\*)"running";

if(p->rest <= 0) { // 该进程已经执行完

m1 = m1 - slice + temp; // 进程完成时间

round = m1 - p->arrive; // 进程周转时间

weighted\_ = round / (p->run); // 进程带权周转时间

sum\_round += round; // 所有进程周转时间之和

sum\_weighted\_ += weighted\_; // 所有进程带权周转时间之和

p->rest = 0;

}

printf("\n------------------------------------[Total time = %.3f]\n", m1);

printf("name\tarrive\trun\trest\tstate\n");

printf("%c\t%.3f\t%.3f\t%.3f\t%s\n", p->name, p->arrive, p->run, p->rest, p->state);

while(head != NULL && head->arrive <= m1)

{ // 下一个进程的到达时间 <= 当前所花的时间片

t = head;

head = head->next;

t->next = NULL;

InsertQue(t); // 把该进程放入到队列里面

}

if(p->rest > 0){ // 如果当前进程还没有执行完

p->next = NULL;

InsertQue(p); // 则需要把当前没执行完的进程继续放入到队列中

}

}

printf("\n总周转时间： "); printf("%.3f\n", sum\_round);

printf("总带权周转时间： "); printf("%.3f\n", sum\_weighted\_);

avr\_round = sum\_round / a;

avr\_weighted\_ = sum\_weighted\_ / a;

printf("\n平均周转时间： "); printf("%.3f\n",avr\_round);

printf("平均带权周转时间： "); printf("%.3f\n",avr\_weighted\_);

}

**（5）主函数**

**int main()**{

float t1 = 0.0f; // 总周转时间

float t2 = 0.0f; // 总带权周转时间

float avr\_t1 = 0.0f; // 平均周转时间

float avr\_t2 = 0.0f; // 平均带权周转时间

int n, i;

char select = ' '; // 选择算法的变量标识

while(select != 'e' && select != 'E'){ // 不是退出标识，保持循环

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

printf("请选择算法：\na. FCFS\_SPN \nb. SPN \nc. RR \ne.exit \n输入选择的字母：");

scanf("%c", &select);

if(select == 'a' || select == 'A') // 选择 FCFS\_SPN 算法{

printf("\n\n=========先来先服务算法FCFS\_SPN==========\n\n");

printf("请输入进程数数量num：");

scanf("%d", &n);

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

printf("%d 进程名name：", i+1); scanf("%s", &a[i].name);

printf("到达时间arrivetime："); scanf("%f", &a[i].arrivetime);

printf("服务时间servetime："); scanf("%f", &a[i].servetime);

}

sortarrivetime(a, n); // 先对到达时间从小到大冒泡排序

FCFS\_SPN(a, n, t1, t2); // 调用先来先服务算法

avr\_t1 = t1 / n;

avr\_t2 = t2 / n;

printf("\n");

printf("平均周转时间avrRoundtime为：%.3f \n", avr\_t1);

printf("平均带权周转时间avrWeighted\_time为：%.3f \n", avr\_t2);

}

else if(select == 'b' || select == 'B') { // 选择 SPN 算法

printf("\n\n===========短进程优先算法SPN============\n\n");

printf("请输入进程数数量num：");

scanf("%d", &n);

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

printf("%d 进程名name：", i+1); scanf("%s", &a[i].name);

printf("到达时间arrivetime："); scanf("%f", &a[i].arrivetime);

printf("服务时间servetime："); scanf("%f", &a[i].servetime);

}

sortservetime(a, n); // 先对服务时间从小到大冒泡排序

SPN(a, n, t1, t2); // 调用短进程优先算法

avr\_t1 = t1 / n;

avr\_t2 = t2 / n;

printf("\n");

printf("平均周转时间avrRoundtime为：%.3f \n", avr\_t1);

printf("平均带权周转时间avrWeighted\_time为：%.3f \n", avr\_t2);

}

else if(select == 'c' || select == 'C') { // 选择 RR 算法

printf("\n\n============时间片轮转算法RR===========\n\n");

int a;

printf("请输入进程数数量num："); scanf("%d", &a);

struct RR \*head;

head = create(a);

RR(head, a); // 调用时间片轮转算法

}

t1 = 0; t2 = 0;

getchar();

}return 0;

}

2、计算每种算法下的平均周转时间、平均带权周转时间

**（1）平均周转时间、平均带权周转时间的计算公式如下：**

进程周转时间 ＝ 进程完成时间 － 进程提交时间

平均周转时间 ＝ 所有进程的周转时间之和 / 进程数

带权周转时间 ＝ 进程周转时间 / 进程运行时间

平均带权周转时间 ＝ 所有进程的带权周转时间之和 / 进程数

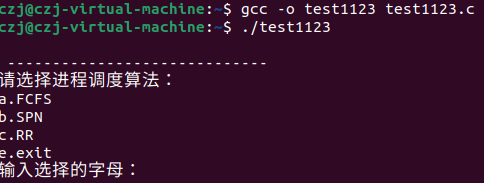
在程序实现的过程中，先计算各个进程的周转时间（以及带权周转时间），同时求各周转（带权周转）时间的和，最后除以总的进程数，得到平均周转时间（以及平均带权周转时间）。

**（2）在上机操作与运行结果截图：**

·打开终端，创建c文件，然后进入vi编辑器，编辑该文件；

·按下键盘“i”键，编写 C 语言代码，编辑完成后按 esc 键，输入:wq 保存并退出；

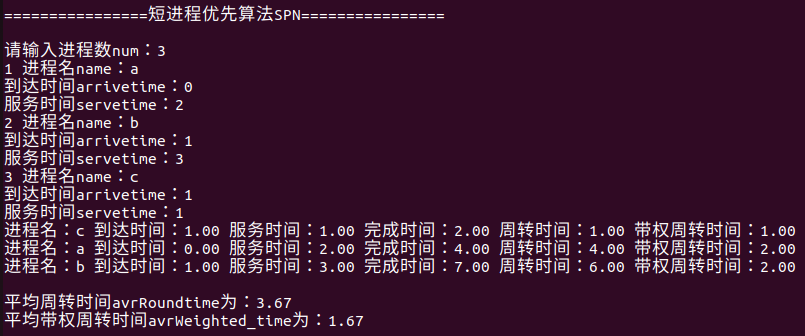
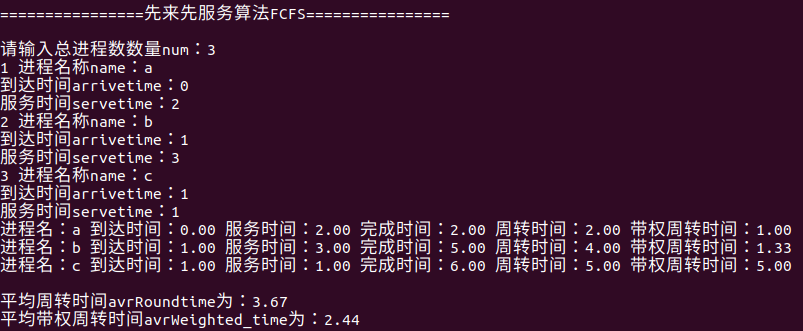
·编译链接 文件，然后运行该文件，运行截图如下；

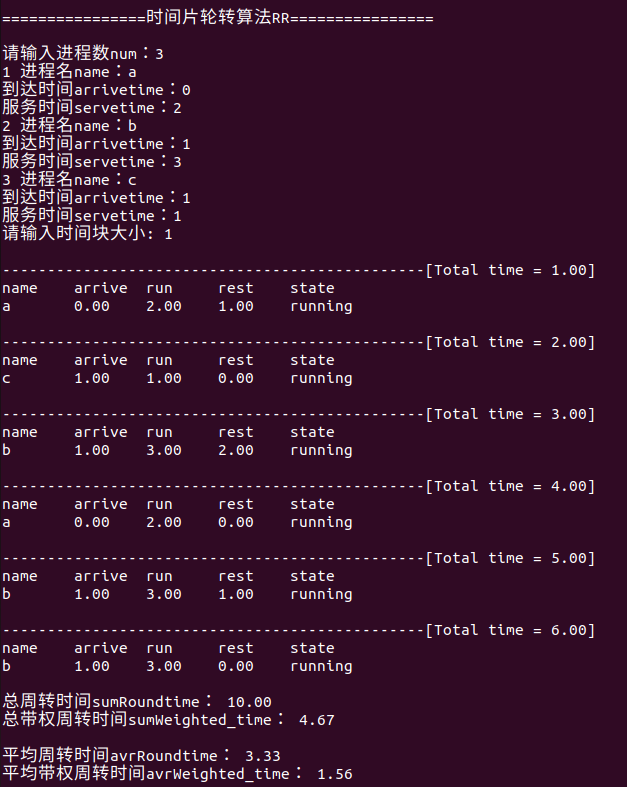


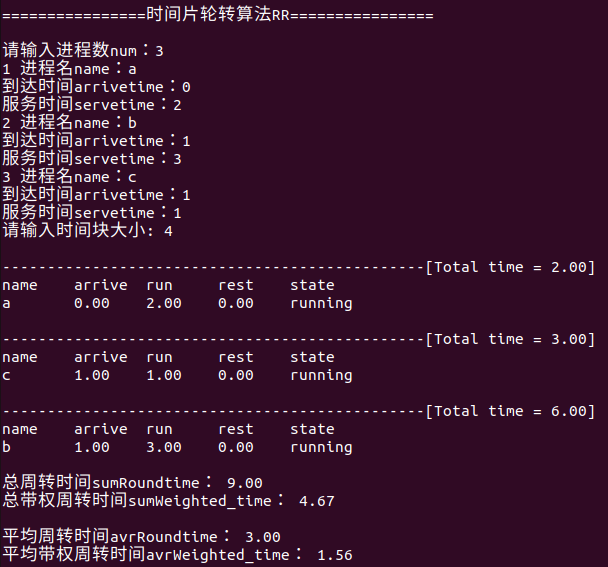
|  |  |  |
| --- | --- | --- |
| 输入进程数：3 | | |
| 进程名 | 到达时间 | 服务时间 |
| a | 0 | 2 |
| b | 1 | 3 |
| c | 1 | 1 |

·输入示例：

各算法实现过程截图如下：



-- 时间片为1： -- 时间片为 4：



综上，以上述的输入为例，每种算法下的平均周转时间、平均带权周转时间的计算结果汇总如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | FCFS\_SPN | SPN | RR（时间片1） | RR（时间片4） |
| 平均周转时间 | 3.67 | 3.67 | 3.33 | 3.00 |
| 平均带权周转时间 | 2.44 | 1.67 | 1.56 | 1.56 |

# 四．思考题

1、在实际的进程调度中，除了按调度算法选择下一个执行的进程外，还应该处理哪些工作？

（1）保存处理机的现场信息。在进程调度时，首先需要保存当前进程的处理机的现场信息，如：程序计数器、多个通用计数器中的内容等。

（2）在按调度算法选择好进程后，由分派程序把处理机分配给该进程，此时需要将选中进程的PCB内有关处理机现场的信息装入处理器相应的各个寄存器中，把处理器的控制权交予该进程，让它从上次的断点处恢复运行。

2、几种进程调度算法有何区别与联系？

（1）FCFS\_SPN

先来先服务调度算法是按照进程/作业到达的先后次序分配CPU，即优先考虑在系统中等待时间最长的进程/作业；一般采用非抢占的策略。既可用于进程调度，也可用于作业调度；有利于长进程，不利于短进程；有利于 CPU 繁忙的进程，不利于 I/O 繁忙的进程。

（2）SPN

短进程优先的调度算法是以作业/进程的长短来计算优先级，越短优先级越高，而作业/进程的长短以服务时间来衡量；一般也采用非抢占的策略。既可用于进程调度，也可用于作业调度；与 FCFS\_SPN 算法相比，可改善平均周转时间和平均带权周转时间，缩短进程的等待时间，提高系统的吞吐量；但对长进程非常不利，可能长时间得不到执行，难以准确估计进程的执行时间，从而会影响调度性能。

（3）RR

时间片轮转调度算法是让每个进程在就绪队列中的等待时间与享受服务的时间能够正相关，将系统中所有的就绪进程按照 FCFS\_SPN 规则，排成一个队列，设置每隔一定的时间间隔就产生一次中断，调度下一个队首进程；所以一般是采用抢占式的策略。一般只用于进程调度，最常应用于分时系统中，能保证及时地响应用户的请求。

3、RR 调度算法中，如果将时间定量增长为一个任意大的数目，那么会产生什么影响？

如果时间片增长为一个过大的数目，大到能够使每个进程都能在一个时间片内完成，那么RR算法也就退化为了FCFS\_SPN算法，无法满足短作业和交互式用户的需求了。