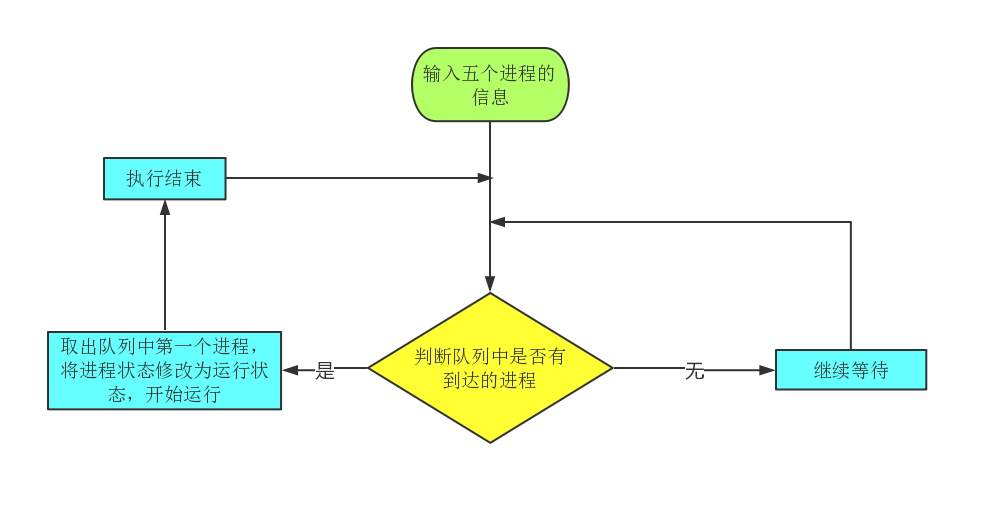
处理机调度实验

一、先来先服务调度算法：

一、流程图



二、源程序

#include<stdio.h>

#include<stdlib.h>

//输入5个进程的信息

struct process{

char name[10]; //进程名

int state; //进程状态

int arriveTime; //到达时间

int runTime; //运行时间

int beganTime; //开始运行时间

int estimatedTime;

struct estimatedTime thread \*next;

};

//将输入的5个进程按照到达的时间排序

int sort(struct process array[5]){

int i,j;

struct process temp = array[0];

for(j = 0;j < 5;j++){

for(i = 0;i < 5-j-1;i++){

if(array[i].arriveTime > array[i+1].arriveTime){

temp = array[i+1];

array[i+1] = array[i];

array[i] = temp;

}

}

}

return 0;

}

int increaseTime(int timeCounter,struct process array[5]){

int i;

sleep(1);

timeCounter++;

for(i = 0;i < 5;i++){

//判断当前时刻是否有作业到达

if(array[i].arriveTime == timeCounter){

printf("thread %s prepared,arriveTime is %d\n",array[i].name,timeCounter);

}

}

return timeCounter;

}

int main(){

struct process p[5];

int i;

int timeCounter = 0;

int estimatedTime;

int avgRunT = 0;

for(i = 0;i < 5;i++){

printf("input the process name\n");

scanf("%s",&p[i].name);

printf("input the process estimateTime\n");

scanf("%d",&p[i].estimatedTime);

printf("input the process arriveTime\n");

scanf("%d",&p[i].arriveTime);

}

sort(p);

//显示刚刚输入的进程的相关信息

for(i = 0;i < 5;i++){

printf("process name: %s estimatedTime: %d arriveTime: %d\n",p[i].name,p[i].estimatedTime,p[i].arriveTime);

}

printf("\n\n");

for(i = 0;i < 5;i++){

//等待进程到来

while(p[i].arriveTime > timeCounter){

timeCounter = increaseTime(timeCounter,p);

}

p[i].beganTime = timeCounter;

estimatedTime = p[i].estimatedTime;

printf("process %s began run,began time %d\n",p[i].name,timeCounter);

//执行当前进程

while(estimatedTime--){

timeCounter = increaseTime(timeCounter,p);

}

printf("process %s is end,end time %d\n",p[i].name,timeCounter);

}

printf("\n\n");

// 打印每个进程的开始运行时间，运行结束时间，周转时间

for(i = 0;i < 5;i++){

avgRunT += p[i].beganTime + p[i].estimatedTime - p[i].arriveTime;

printf("process name:%s began run time %d,end time %d,Runperiod time%d \n",p[i].name,p[i].beganTime,p[i].beganTime + p[i].estimatedTime,p[i].beganTime + p[i].estimatedTime - p[i].arriveTime);

}

printf(" average Runperiod time is %d\n",avgRunT/5);//打印平均周转时间

return 0;

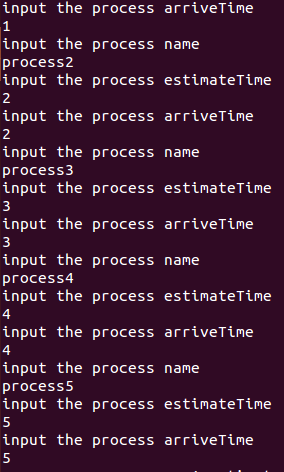
}

三、程序运行环境、使用工具说明

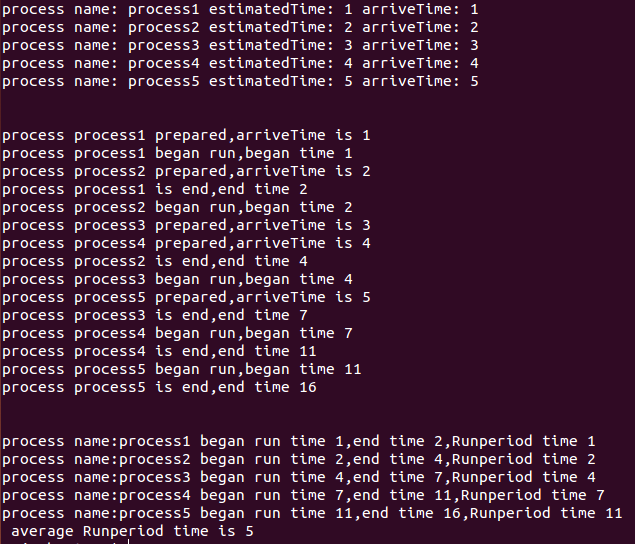
Linux环境下，VMware工具

四、程序运行结果

输入的5个进程：



输出：

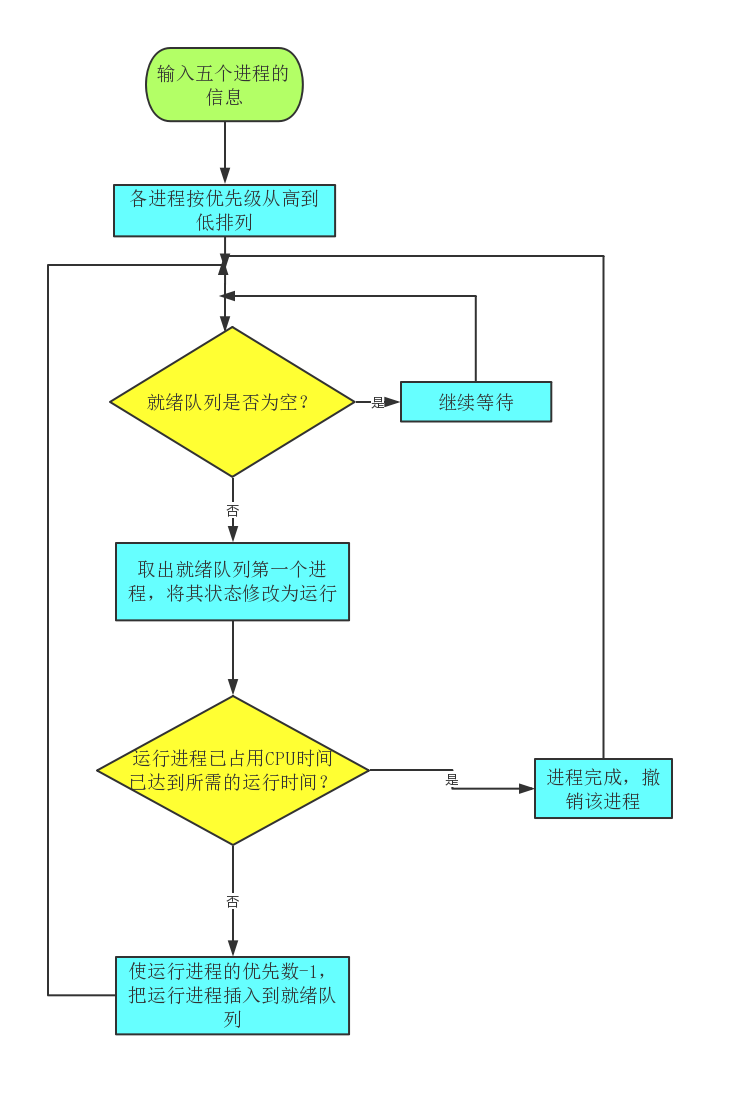


五、结果的简单说明或分析

先来先服务调度算法就是每次调度从就绪队列中选择一个最先进入该队列的进程进程处理。

一、基于优先级的调度算法：

一、流程图



二、源程序

#include <stdio.h>

#include <stdlib.h>

#define getpch(type) (type\*)malloc(sizeof(type))

//#define NULL 0

//定义PCB结构体

struct pcb

{

char name[10]; //进程名称

int state; //进程状态

int super; //优先级

int ntime; //总运行时间

int rtime; //已运行时间

struct pcb\* link; //下一PCB的地址

} //定义3个pcb类型指针,其中:

\*ready=NULL, //代表就绪队列链表首位地址

\*p, //代表当前正在运行的pcb地址

\*bhead=NULL; //代表阻塞链表的首位地址

//把结构体pcb定义成一个数据类型:PCB

typedef struct pcb PCB;

//排序函数:对就绪队列里面的进程根据优先级进行排序

void sort()

{

PCB \*first,\*second; //没什么好说的

int insert=0; //该值用于标记在排序过程中是否发生过插队情况

//如果[就绪链表首项为空]或[当前进程的优先数高于链表首项进程的优先数]

if ((ready==NULL)||((p->super)>(ready->super)))

{

//将当前进程置于就绪链表首位

p->link=ready;

ready=p;

}

else

{

//取出链表中的排在最前两项,分别为first,second

first=ready;

second=first->link;

while(second!=NULL)

{

//如果p的优先级大于second

if((p->super)>(second->super))

{

//将p插于first和second之间

p->link=second;

first->link=p;

second=NULL; //设置second为空(这是循环结束的条件)

insert=1; //将的值"insert"设置为1

}

else //否则 0.0

{

first=first->link;second=second->link;

}

}

//如果没有发生过插队情况

if(insert==0) {

first->link=p; }

}

}

//阻塞函数:阻塞就绪队列中(从前往后)优先级相同的连续进程

void block(){

PCB \*pb,\*index=ready;

int count=0;

if(ready==NULL)return; //如果就绪队列中没有进程,则终止阻塞操作(都没有进程我还阻塞个啥?)

//循环,遍历整个就绪链表

while((index->link!=NULL))

{

if((index->link->super)>=ready->super){

index->link->state='B'; //将进程状态标记为'B',即"block".

index=index->link; //用以链表循环的循环指针向后移一位.

count++; //每检测到可阻塞进程时自增计数

}else{

break;

}

}

//判断当就绪链表中可阻塞进程大于0时

if(count>0){

pb=index->link;

index->link=bhead;

bhead=ready->link;

ready->link=pb;

}

}

//唤醒函数:将阻塞队列中优先数最大且剩余运行时间最小的进程调入就绪链表首位

void weakup(){

PCB \*index,\*prior,\*t;

int exchange=0; //该值用于标记是否需要进行位置调换操作

if(bhead==NULL)return; //如果就阻塞列中没有进程,则终止唤醒操作(都没有进程我还唤醒个啥?)

//如果阻塞链表不止一个进程

if(bhead->link!=NULL)

{

index=bhead->link;

prior=bhead;

//循环,遍历整个阻塞链表

while(index!=NULL)

{

if((index->super)>(bhead->super))

{

exchange=1; //标记需要进行位置调换操作

}

else if((index->super)==(bhead->super))

{

if((index->ntime-index->rtime)<(bhead->ntime-bhead->rtime))

{

exchange=1; //标记需要进行位置调换操作

}

}

if(exchange==1){

//与阻塞链表首位进程调换位置

prior->link=bhead;

t=bhead->link;

bhead->link=index->link;

index->link=t;

bhead=index;

//将标记变量exchange重新标记为0

exchange=0;

}

prior=index; //"prior"向后移动一位(这里之所以不使用"prior=prior->link"是因为执行交换位置操作时,prior的后继地址将不为index)

index=index->link; //"index"向后移动一位

}

}

//将阻塞链表首位的进程调到就绪列表首位

t=bhead->link;

bhead->link=ready;

ready=bhead;

bhead=t;

}

//录入进程信息

void input()

{

int i,num;

printf("\n请输入进程数量:");

scanf("%d",&num);

//循环录入每个进程的信息

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

{

//赋予每个进程初始属性值

printf("\n[请设置 %d号 进程的基本属性]\n",i+1);

p=getpch(PCB); //getpch(type)是宏定义函数,使用malloc函数根据数据类型,动态开辟内存空间

printf("\n 名称：");

scanf("%s",p->name);

printf("\n 优先级：");

scanf("%d",&p->super);

printf("\n 所需时间：");

scanf("%d",&p->ntime);

printf("\n");

p->rtime=0;

p->state='W'; //'W'等待状态,'R'运行状态,'B'阻塞状态

p->link=NULL;

sort(); //执行排序

}

}

//获取链表长度

int space()

{

int l=0;PCB\*pr=ready;

while(pr!=NULL)

{

l++;

pr=pr->link;

}

return(l);

}

void show()

{

printf("\n名称\t状态\t优先数\t总时长\t已用时\n");

}

//显示指定进程的信息

void disp(PCB\*pr)

{

printf("%s\t",pr->name);

printf("%c\t",pr->state);

printf("%d\t",pr->super);

printf("%d\t",pr->ntime);

printf("%d\t",pr->rtime);

printf("\n");

}

//显示当前正在运行的进程和就绪队列内进程的信息

void check()

{

PCB \*pr,\*pb;

printf("\n====[当前运行中的进程]====\n");

printf("进程名称:%s\n",p->name);

show(); //显示列名

disp(p);//显示当前正在运行的进程信息

pr=ready;

if(pr==NULL){

printf("\n====[当前就绪队列为空]====\n");

}

else

{

printf("\n====[当前就绪队列状态]====");

show();

while(pr!=NULL)

{

disp(pr);

pr=pr->link;

}

}

pb=bhead;

if(pb==NULL){

printf("\n====[当前阻塞队列为空]====\n");

}

else

{

printf("\n====[当前阻塞队列状态]====");

show();

while(pb!=NULL)

{

disp(pb);

pb=pb->link;

}

}

}

//销毁函数:销毁当前正在运行的进程

void destroy()

{

printf("\n进程[%s]已完成.\n",p->name);

free(p);

}

//运行函数

void running()

{

(p->rtime)++; //当前进程的已运行时间自增

//如果进程实际运行时间已经等于了预设的总时长

if(p->rtime==p->ntime)

{

destroy();

}

else

{

(p->super)--; //进程优先数减一

p->state='W'; //状态标记为等待

sort(); //就绪队列重新排序

//如果阻塞链表不为空则唤醒阻塞进程

if(bhead!=NULL){

weakup();

}

block(); //执行阻塞操作

}

}

//主函数

void main()

{

int len,h=0;

char ch;

input(); //录入进程信息

len=space(); //获取链表长度(没啥o用)

//只要链表长度不等于0且链表首项不为空就一直循环下去

while((len!=0)&&(ready!=NULL))

{

//system("cls");

ch=getchar();

h++;

printf("\n\n[第%d次运行]\n",h);

//取就绪链表的首项进程,转为正在进行状态

p=ready;

ready=p->link;

p->link=NULL;

p->state='R';

check(); //显示当前运行状态信息

running(); //运行

printf("\n按任一键继续.....");

ch=getchar();

}

printf("\n\n进程已经完成.\n");

ch=getchar();

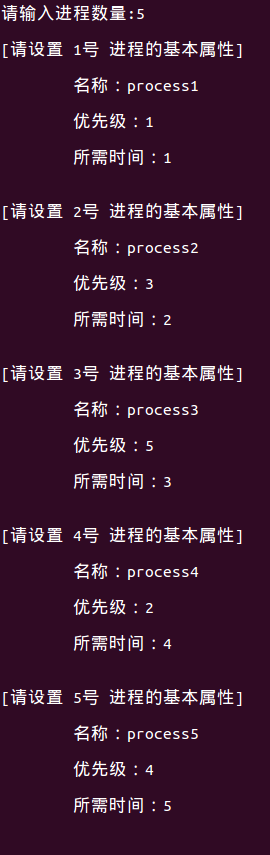
}

三、程序运行环境、使用工具说明

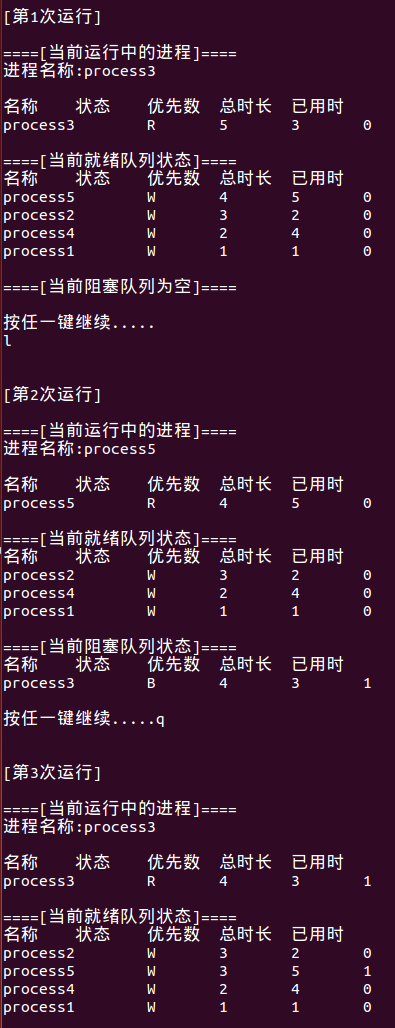
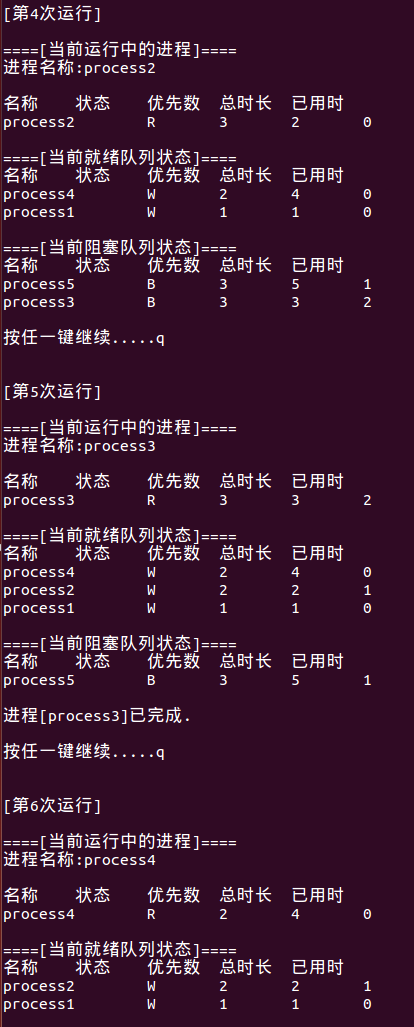
Linux下，Vmware工具

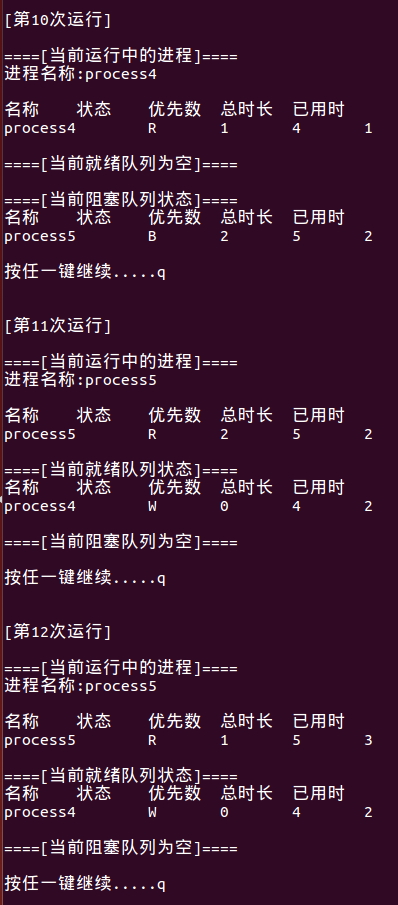
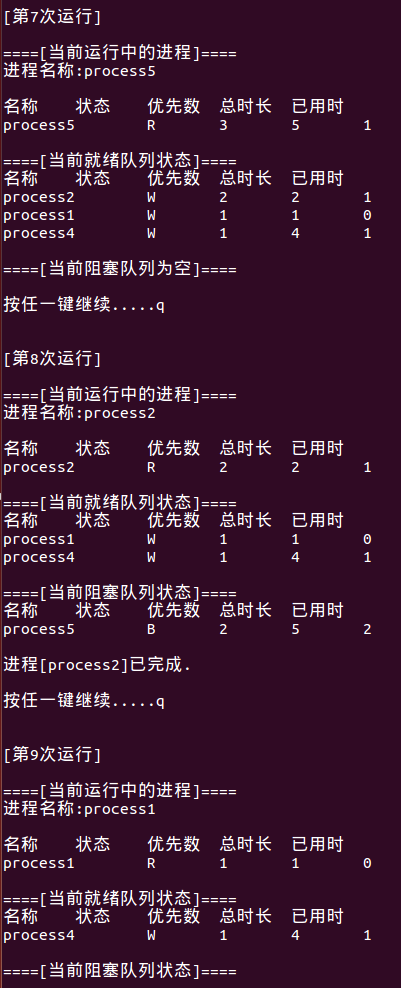
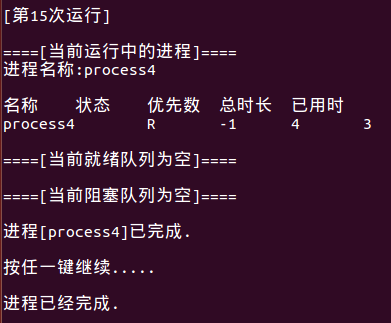
四、程序运行结果

输入：



输出：

五、结果的简单说明或分析

基于优先级的调度算法是把处理机分配给就绪队列中优先级最高的进程，可分为抢占式和非抢占式。每次先将输入的5个进程通过优先级进行排序。