****

**实 验 报 告**

学院： 计算机科学学院 专业： 计算机科学与技术 2023年5月6日

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 姓 名 |  | 学 号 |  | |
| 班 级 |  | 指导老师 |  | |
| 课程名称 | 操作系统原理实践 | | 成  绩 |  |
| 实验名称 | 进程调度算法的模拟 | |
| 1. 实验目的 | | | | |
| 1. 实验内容 | | | | |
| 1. 实验环境   VMware Workstation Pro虚拟机，Ubuntu 64位 | | | | |
| 1. 实验方法和步骤（含设计）   具体调度算法：FCFS、SJF、PR  涉及多种操作：排序、链表操作  模拟三种调度算法程序代码  #include<stdio.h>  #include<stdlib.h>  #include<string.h>  #include<stdbool.h>  typedef struct node{//定义结构体node，别名是进程控制块PCB，里面存储了进程的一些信息，也是创建链表，定义链表的数据域和指针域，typedef目的是给结构体设置别名  char name[10];//进程号  int prio;//优先数  int round;//时间片  int cputime;//cpu时间  int needtime;//所需时间  int count;//记数  char state;//状态  struct node \*next;//指针域  }PCB;  PCB \*finish,\*ready,\*run,\*r;//定义进程控制块PCB的头结点finish、就绪队列ready、正在运行的进程run和指向就绪队列的指针r  int N;//定义全局变量N，进程数目  void firstin()//将就绪队列中的第一个进程作为当前运行进程  {  run = ready;//run指向ready所在位置  run->state = 'R';//状态改为运行  ready = ready->next;//ready指向下一个链表项  }  void prt1(char a)//输出PCB信息的辅助函数  {  if(a=='P'||a == 'p')  printf("进程号 cpu时间 所需时间 优先数 状态\n");//a为P或p时输出进程信息  else if(a =='r'||a =='R')  printf("进程号 cpu时间 所需时间 记数 时间片 状态\n");//a为R或r时输出时间片信息  else if(a=='f'||a=='F')  printf("进程号 所需时间 状态\n");//a为F或f时输出结束信息  }  void prt2(char a ,PCB \*q)//输出PCB信息的函数  {  if(a=='P'||a == 'p')  printf("%-10s%-10d%-10d%-10d%c\n",q->name,q->cputime,q->needtime,q->prio,q->state);//a为P或p时输出进程信息  else if(a == 'r'|| a == 'R')  printf("%-10s%-10d%-10d%-10d%-10d%c\n",q->name,q->cputime,q->needtime,q->count,q->round,q->state);//a为R或r时输出时间片信息  else if(a=='f'||a=='F')  printf("%-10s%-10d%c\n",q->name,q->needtime,q->state);//a为F或f时输出结束信息  }  //输出所有进程的信息 ，P或p表示优先级调度，R或r表示时间片轮转调度，F或f表示先来先服务调度  void prt(char algo)//algo为调度算法类型  {  PCB \*p;  prt1(algo);  if(run != NULL&& !(algo == 'r'||algo =='R'))//这个对RR不适应！！  prt2(algo,run);//输出运行进程信息  p = ready;  while(p != NULL)//输出所有就绪进程信息  {  prt2(algo,p);  p = p->next;  }  p = finish;  while(p != NULL)//输出所有结束进程信息  {  prt2(algo,p);  p = p->next;  }  getchar();  }  //在就绪队列中按优先级插入一个新进程  void insert1(PCB \*q)//定义函数，入参为指向PCB的指针  {  PCB \*p1,\*s,\*r;//定义指向PCB的指针变量 p1,s,r  int b; //定义变量 b  s = q;//将入参传入为新建的s PCB  p1 = ready; //将当前就绪队列的头进程赋值给p1，即从头开始遍历  r = p1;//将p1赋值给r，作为待插入PCB所要插入的位置  b = 1; //将 b 赋值为 true，也即是走while循环  while((p1 != NULL)&&b)//找到合适的插入位置  {  if(p1->prio >= s->prio) //如果当前进程优先级大于等于新进程优先级  {  r = p1;  p1 = p1->next;  }  else b = 0; //将 b 赋值为 false， 即结束while循环  }  if(r!= p1)//如果待插入PCB不是插在队列头，指针发生变化，需要将到达的位置插入新PCB  {  r->next = s;//将位置r的下一个指针赋值为待插入的PCB  s->next = p1;//将待插入PCB的下一个指针赋值为位置p1  }  else //如果插入队列头部  {  s->next = p1; //将待插入PCB的下一个指针赋值为p1  ready = s; //就绪队列头部赋值为待插入PCB，即将PCB插到就绪头部  }  }  //创建进程链表  void creat1(char alg)  {  PCB \*p;//定义指向进程控制块的指针p  int i,time;//定义循环计数变量i、时间变量time  char na[10];//定义进程名数组na  ready = NULL; //就绪队列置为空  finish = NULL;//结束队列置为空  run = NULL;//运行进程置为空  printf("输入进程号和运行时间:\n");  for(i = 1; i <= N; i++) //循环输入进程信息  {  p = (PCB \*)malloc(sizeof(PCB));//为进程控制块分配空间  scanf("%s",na);//进程名  scanf("%d",&time);//运行时间  //设置进程控制块的各项参数  strcpy(p->name,na); //将进程名复制到进程控制块中  p->cputime = 0;//CPU时间初始化为0  p->needtime = time; //设置需要运行的时间  p->state = 'W';//将进程状态置为等待  p->prio = 50-time; //设置进程的优先级，优先数越小优先级越高  if(ready != NULL)//如果就绪队列非空，则将进程插入到就绪队列中  insert1(p); //优先数越小优先级越高，将进程插入就绪队列中  else{  p->next = ready; //将当前进程放在就绪队列的第一个位置  ready = p;  }  }  //clrscr();  //根据算法类型输出不同的信息  if(alg =='p'||alg =='P')  printf(" 优先数算法输出信息:\n");  else  printf(" 先来先服务算法输出信息:\n");  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  prt(alg);//终端打印就绪队列中的进程信息  run = ready;//将运行进程设为就绪队列的第一个进程  ready = ready->next;//将就绪队列的第一个进程删除（出队），使其成为下一个运行进程的备选  run->state = 'R';//将当前运行进程的状态设置为“运行”  }  //函数定义：创建进程  void creat2(char alg)  {  //定义指向进程控制块的指针p，以及循环计数变量i、时间变量time和进程名数组na  PCB \*p;  int i,time,round;  char na[10];  //初始化就绪队列、结束队列和运行进程为空  ready = NULL;  finish = NULL;  run = NULL;  printf("请输入时间片：");  scanf("%d",&round);  printf("输入进程号和运行时间:\n");  for(i = 1; i <= N; i++)//循环输入进程信息  {  p = (PCB \*)malloc(sizeof(PCB)); //为进程控制块分配空间  scanf("%s",na); //输入进程名  scanf("%d",&time);//输入需要运行的时间  strcpy(p->name,na);//将进程名复制到进程控制块中  p->cputime = 0; //CPU时间初始化为0  p->needtime = time;//设置需要运行的总时间  p->state = 'W';//将进程状态置为等待状态  p->count = 0;// 设置已运行时间片数为0  p->round = round;//设置时间片大小  p->next = NULL;  if(i == 1)//按顺序插入到ready链表中  r = ready = p;  else  r->next = p;//将当前进程添加到链表末尾  r = p; //将r指向新的链表末尾  }  //clrscr();  printf(" 时间片轮转算法输出信息:\n");  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  prt(alg);//终端打印就绪队列中的进程信息  }  void priority(char alg) //优先数调度算法  {  while(run != NULL)//只要当前有运行进程，就一直循环  {  run->cputime += 1; //当前运行进程的CPU时间加1  run->needtime -= 1; //当前运行进程的剩余运行时间减1  run->prio -= 3; //当前运行进程的优先级减3  if(run->needtime == 0)//如果当前运行进程的剩余运行时间为0，说明该进程已经执行完毕  {  run->next = finish;//将其添加至结束队列的队尾  finish = run;  run->state = 'F';//将其状态设为“完成”  run = NULL;//将运行进程设为空  if(ready != NULL)//如果就绪队列非空，则选取优先级最高的进程作为运行进程  firstin();  }  else  if((ready != NULL) && (run->prio < ready->prio))//如果当前进程的剩余时间不为0，且就绪队列非空  {  run->state = 'W'; //将当前进程的状态设为“等待”  insert1(run);//将当前进程加入到就绪队列，按照优先级从高到低排序  firstin(); //选取优先级最高的进程作为运行进程  }  prt(alg);//打印当前就绪队列中的进程信息  }  }  void roundrun(char alg) //时间片轮转调度算法  {  bool flag;//当ready列里只有一个时做标记  while(N){  flag = 1;//初始化为1  run = ready;//run每次运行ready的队头  run->count++;//每运行一次计数器加1  if(run->needtime < run->round)//当剩余时间小于时间片轮转时间时的情况  {  run->cputime += run->needtime;  run->needtime = 0;  }  else{  run->cputime += run->round;  run->needtime -= run->round;  }  run->state = 'W';//变为等待  if(ready->next != NULL)  ready = ready->next;  else flag = 0;//当ready剩一个时做标记  if(run->needtime == 0){//当run结束时放入finish队列里  run->next = finish;  finish = run;  run->state = 'F';  N--;//进程数少1  }  else{  if(flag){//执行完如果不是剩一个的话，就把run放到队尾  r->next = run;  r = run;  r->next = NULL;  }  }  if(N)ready->state = 'R';//结束时不应该有'R"  else  ready = NULL;//结束时应该为空  prt(alg);//输出  }  }  void FCFSrun(char alg) //先来先服务调度算法  {  PCB \*p;//定义指向进程控制块的指针p  while(run!=NULL)//只要当前有正在运行的进程，就一直循环  {  run->cputime += run->needtime; //将当前进程的CPU时间加上其需要的总时间  run->needtime =0; //将当前进程的剩余运行时间设为0  run->next = finish;//将当前进程添加至结束队列的队尾  finish = run; //更新结束队列  run->state = 'F';//将当前进程的状态设为“完成”  run = NULL;//将运行进程设为空  if(ready!=NULL)//如果就绪队列非空，则选取队首进程作为运行进程  firstin();  prt(alg);//打印当前就绪队列中的进程信息  }  }  main()  {  char algo;  //clrscr();  printf("选择算法:P/R/F(优先数算法/时间片轮转算法/先来先服务算法)\n");  scanf("%c",&algo);//输入进程数目  printf("输入进程数:\n");  scanf("%d",&N);//读入用户输入的进程数目  //根据算法类型选择不同的进程调度算法  if(algo == 'P'||algo == 'p') //优先数调度算法  {  creat1(algo);//创建进程链表  priority(algo); //运行优先数调度算法  }  else if(algo == 'R'||algo == 'r')//时间片轮转调度算法  {  creat2(algo); //创建进程链表  roundrun(algo);//运行时间片轮转调度算法  }  else if(algo=='f'||algo=='F') //先来先服务调度算法  {  creat1(algo);//创建进程链表  FCFSrun(algo); //运行先来先服务调度算法  }  system("echo 按任意键退出");  system("read -n 1");  return 0;//程序执行完毕，返回0  }  先来先服务算法流程图  wps   1. 结合时间片轮转调度算法程序代码   #define \_CRT\_SECURE\_NO\_WARNINGS  #include "stdio.h"  #include <stdlib.h>  #include <conio.h>  #define getpch(type) (type\*)malloc(sizeof(type)) //快速malloc  #define NULL 0  #define q 1 //时间片  struct pcb { /\* 定义进程控制块PCB \*/  char state;//进程状态 就绪 W（Wait）、运行R（Run）、或完成F（Finish）  int Atime=0;//到达时间，默认同一时间到达  int ntime;//进程运行时间  int rtime;//已用CPU时间  int TF;//是否为就绪队列内的有效PCB，默认有效  struct pcb\* link;  //新建变量：是否到达的一个状态  char name[10];//进程名  }\*ready = NULL, \* p;  typedef struct pcb PCB;  int h = 0;//执行时间片数  int Time = 0;//CPU运行时间 利用Time记录每个进程的完成时间 (Time-Atime) 是周转时间  float allAvgTime = 0;//累加所有进程的带权周转时间  /\* 当该进程的时间片耗尽或运行完毕时，将该进程插入有效就绪队列\*/  void roundrun()  {  PCB\* first, \* second;  if (ready == NULL||ready->TF==0) /\*就绪队列为空，或者进程重新插入时，就绪队列有效PCB为0，插入队首\*/  {  p->link = ready;  ready = p;  }  else /\*尾插 只能插入到有效PCB的后面\*/  {  first = ready;  second = first->link;  while (second != NULL&&second->TF==1)  {  first = first->link;  second = second->link;  }  p->link = second;  first->link = p;  }  }  /\*PCB必须按顺序输入，到达时间从小到大。\*/  /\* 建立进程控制块函数，按顺序插入到就绪队列中\*/  void input()  {  int i, num;  system("cls"); /\*清屏\*/  printf("\n 请输入建立的进程数?");  scanf("%d", &num);  for (i = 0; i < num; i++)  {  printf("\n 进程号No.%d:\n", i);  p = getpch(PCB);//申请结构体PCB内存空间  printf("\n 输入进程名:");  scanf("%s", &p->name);  printf("\n 输入进程到达时间:");  scanf("%d", &p->Atime);  printf("\n 输入进程运行时间:");  scanf("%d", &p->ntime);  p->link = NULL;  p->state = 'w';  p->rtime = 0;  p->TF = 1;  printf("\n");  roundrun(); //调用sort函数，插入p进程进入ready队列  }  }  /\*查看就绪队列中有多少进程\*/  int space()  {  int l = 0; PCB\* pr = ready;  while (pr != NULL)  {  l++;  pr = pr->link;  }  return(l);  }  /\*建立进程显示函数,用于打印当前进程\*/  void disp(PCB\* pr)  {  printf("\n qname \t state \t Atime \t ndtime runtime daoda(0/1) \n");  printf("|%s\t", pr->name);  printf("|%c\t", pr->state);  printf("|%d\t", pr->Atime);  printf("|%d\t", pr->ntime);  printf("|%d\t", pr->rtime);  printf(" |%d\t", pr->TF);  printf("\n");  }  /\* 建立进程查看函数 \*/  void check()  {  PCB\* pr;  printf("\n \*\*\*\* 当前正在运行的进程是:%s", p->name); /\*显示当前运行进程\*/  disp(p);//封装  pr = ready;  printf("\n \*\*\*\*当前就绪队列状态为:\n"); /\*显示就绪队列状态\*/  while (pr != NULL)  {  disp(pr);//调用打印  pr = pr->link;  }  }  /\*建立进程撤消函数(进程运行结束,撤消进程)\*/  void destroy()  {  allAvgTime = allAvgTime + (float)(Time - p->Atime) / p->ntime;  printf("\n 进程 [%s] 已完成.\n周转时间为[%d ms]\n带权周转时间为[%.2f ms]\n", p->name,Time-p->Atime,(float)(Time - p->Atime)/p->ntime);  free(p);  }  /\*在每次调用running中的sort前，刷新PCB的状态（是否到达）\*/  void flushed() {  PCB\* traverse, \* pre;  traverse = ready;  pre = NULL;  while (traverse != NULL&&traverse->TF==1) {  //只需将未到达的PCB置为0即可  if (traverse->Atime > Time) {//不应该是finish时间，应该是CPU执行时间(h\*q 比CPU执行时间略大 暂时就这样吧)  traverse->TF = 0;  }  //新到达的程序(数组)放就绪队列队头  //记录（front） 记录第一个0-->1（left） 循环遍历到最后一个0->1（rigth） 记录后面链表（behind pre->behind)  //再将ready置为（left）  pre = traverse;  traverse = traverse->link;  }  //本程序只考虑一次改变一个PCB状态  if (traverse != NULL&& traverse->Atime <= Time) {  traverse->TF = 1;  if (pre != NULL) {  pre->link = traverse->link;  traverse->link = ready;  ready = traverse;  }  }  }  /\* 建立进程就绪函数(进程运行时间到,置就绪状态）\*/  void running()  {  (p->rtime)= (p->rtime)+q;  if (p->rtime >= p->ntime) {  if (p->rtime == p->ntime) {  Time = Time + q;  }  else {  Time = Time + (p->ntime) % q;//未利用完时间片，完成跳出  }  p->state = 'F';  //新建一个周转时间（同一时刻创建，所以周转时间等于完成时间） ，记录完成时间  // 根据周转时间/服务时间=带权周转时间  //如果刚好是时间片的整数倍，直接 当前时间片\*时间片数  //如果不是整数倍 则ntime%时间片+时间片  destroy(); /\* 调用destroy函数\*/  }  else  {  Time = Time + q;//未执行完毕  p->state = 'w';//就绪  roundrun(); /\*调用sort函数，重新插入到就绪队列\*/  flushed();//每次都要重新判断是否有PCB到达， 到达直接插入队首.  }  }  int main() /\*主函数\*/  {  int len;  char ch;  input();//建立PCB  len = space();  flushed();//PCB状态刷新  while ((len != 0) && (ready != NULL))  {  ch = getchar();  h++;  printf("\n The execute number:%d \n", h);  if (ready->TF == 0) {//就绪队列内没有有效PCB  //这里只需加上CPU空闲时间即可 第一个无效进程的到达时间-上一个完成进程的完成（注意是完成）时间  //Time= Time+(ready->Atime - Time);  Time = ready->Atime;  //将第一个无效PCB改为有效PCB  ready->TF = 1;  }  else {  p = ready;  ready = p->link;  p->link = NULL;  p->state = 'R';  check();  running();  printf("\n 按任一键继续......");  ch = getchar();  }  }  printf("\n\n 进程已经完成.\n");  printf("系统的平均带权周转时间为【%.2f ms】", allAvgTime / len);  ch = getchar();  }  时间片轮转调度算法流程图  C:/Users/jing_/AppData/Local/Temp/wps.DHLddvwps | | | | | |
| 5．实验分析与体会  实验日期 ： 2023 年 5 月 6 日 | | | | | |
| 教师评语  签名： 年 月 日 | | | | | |