****

**实 验 报 告**

课 程 名 称： 操作系统

实 验 项 目： 时间片轮转调度算法的模拟

专 业 班 级： 23级软件工程专升本3班

姓 名： 罗昕

学 号： 202316060328

日 期： 2023年12月12日

**实践教学中心制**

**实验报告说明**

1. 实验报告应包含实验目的、实验条件、实验方法、实验内容、结果分析、成绩评定等内容。
2. 学生应认真撰写实验内容，并对实验进行总结分析。
3. 指导教师应严格依照成绩评定标准对实验报告进行成绩评定并签字，评语围绕实验报告内容，具体、有针对性、有建设性。
4. 格式要求：A4幅面，双面打印。**封面：**汉字采用三号宋体、外文数字采用三号Times New Roman体，下划线居中填写，下划线整体长度不改变；**正文：**汉字采用小四号宋体、外文数字采用小四号Times New Roman体；左对齐填写，首行缩进2字符，行距固定值20磅，图或表的字体大小为五号字。

|  |  |  |  |
| --- | --- | --- | --- |
| **实验地点：** |  | **指导教师：** |  |
| **同组人员：** |  | | |
| **一、实验目标**  （1）加深对进程概念的理解，明确进程和程序的区别  （2）深入理解系统如何组织进程  （3）理解常用进程调度算法的具体实现 | | | |
| **二、实验条件（环境）**  1.VMware  2.CentOS7 | | | |
| **三、实验方法（原理）**  编写C程序模拟实现单处理机系统中的时间片轮转调度算法，实现对多个进程的调度模拟。  实现方法：  1.用户通过输入设定时间片和各个进程的到达时间和服务时间。  2.使用排序算法，按照进程的到达时间对进程进行排序，以确保进程按照到达时间的先后顺序执行。  3.模拟时间片轮转调度算法的执行。根据剩余服务时间，分情况处理：若剩余服务时间小于等于一个时间片，则该进程执行完成，更新其完成时间、剩余时间，并将其标记为已完成。若剩余服务时间大于一个时间片，则执行一个时间片的任务，并更新剩余服务时间。在每个时间片执行完毕后，检查是否有新的进程到达，若有，将新的进程放入就绪队列中。  4.输出每个进程的详细信息，包括到达时间、服务时间、完成时间、周转时间和带权周转时间。 | | | |
| **四、实验内容（步骤）**  #include <stdio.h> #define MAX\_PROCESSES 100 typedef struct {  int arrival\_time;  int service\_time;  int finish\_time;  int remaining\_time;  int turnaround\_time;  float weighted\_turnaround\_time; } Process; Process processes[MAX\_PROCESSES]; int num\_processes = 0; void sortProcesses() {  int i, j;  Process temp;  for (i = 0; i < num\_processes - 1; i++) {  for (j = 0; j < num\_processes - i - 1; j++) {  if (processes[j].arrival\_time > processes[j + 1].arrival\_time) {  temp = processes[j];  processes[j] = processes[j + 1];  processes[j + 1] = temp;  }  }  } } void roundRobinScheduling(int time\_slice) {  int current\_time = 0;  int i;  for (i = 0; i < num\_processes; i++) {  processes[i].remaining\_time = processes[i].service\_time;  }  while (1) {  bool all\_finished = true;  for (i = 0; i < num\_processes; i++) {  if (processes[i].remaining\_time > 0) {  all\_finished = false;  if (processes[i].remaining\_time <= time\_slice) {  current\_time += processes[i].remaining\_time;  processes[i].finish\_time = current\_time;  processes[i].turnaround\_time = processes[i].finish\_time - processes[i].arrival\_time;  processes[i].weighted\_turnaround\_time = (float)processes[i].turnaround\_time / processes[i].service\_time;  processes[i].remaining\_time = 0;  } else {  current\_time += time\_slice;  processes[i].remaining\_time -= time\_slice;  }  }  }  if (all\_finished) {  break;  }  } } void printProcessInfo() {  printf("进程详情：\n");  for (int i = 0; i < num\_processes; i++) {  printf("进程%d：%d %d %d %d %.2f\n", i + 1, processes[i].arrival\_time, processes[i].service\_time, processes[i].finish\_time, processes[i].turnaround\_time, processes[i].weighted\_turnaround\_time);  }  printf("\n"); } int main() {  int time\_slice;  printf("请输入时间片：");  scanf("%d", &time\_slice);  printf("请输入进程数量：");  scanf("%d", &num\_processes);  for (int i = 0; i < num\_processes; i++) {  printf("请输入进程%d的到达时间和服务时间：", i + 1);  scanf("%d %d", &processes[i].arrival\_time, &processes[i].service\_time);  }  sortProcesses();  roundRobinScheduling(time\_slice);  printProcessInfo();  return 0; } | | | |
| **五、结果分析**  （1）调试过程中遇到的问题以及解决方法，设计与实现的回顾讨论和分析  在算法设计时，由于一开始不知道如何将位于队首的进程，在执行完后如  何移至队尾进行循环，所以思考了很久，后来想到将队首进程进行重新压入队列  从而解决了此问题。  （2）算法的性能分析  每个进程被分配一个时间段，即该进程允许运行的时间。如果在时间片结束  时进程还在运行，则 CPU 将被剥夺并分配给另一个进程。如果进程在时间片结  束前阻塞或结束，则 CPU 当即进行切换。调度程序所要做的就是维护一张就绪  进程列表，当进程用完它的时间片后，它被移到队列的末尾。  （3） 经验体会  通过本次实验，深入理解了时间片轮转 RR进程调度算法的思想，培养了自己的动手能力，通过实践加深了记忆。 | | | |
| **六、成绩评定**  **评 语：**  **评分：**  **指导教师签字：** | | | |