哈尔滨工程大学

实 验 报 告

实 验 名 称：­­

班 级：

学 号：

姓 名：

实 验 时 间：

成 绩：

指 导 教 师：

实验室名称：

**哈尔滨工程大学实验室与资产管理处 制**

|  |
| --- |
| 实验名称：任务调度 |
| 一、问题描述  多用户多任务操作系统中，多个任务同时共享计算机系统资源。为了使多个任务均能够顺利执行，操作系统要按一定的原则对它们进行调度，使它们按一定的次序进行。设只有一个 CPU，现有多个任务，它们需要 CPU 服务的时间已知。在下列假设下，按平均等待时间最短为原则，设计算法求出任务的执行顺序。  1.忽略任务提交的时间差，即认为各任务同时提交。  2.各任务不同时提交  二、数据结构设计  任务调度时，需要执行的任务应该按照一定的次序等待，前面的任务结束后才能执行后面的任务，所以可以用一个队列来存储。这里选用数组存储结构体：  struct task{  int id;  int servicetime;  int starttime;  int finishtime;  int waittime;  };  typedef struct task  三、算法设计  分以下两种情况  1.按提交顺序执行  printf("1.按提交顺序执行:\n");  calculateWaitTime(tasks, n);  2.按最短平均等待时间执行  printf("2.按最短平均等待时间顺序执行:\n");  scheduletasks(tasks, n);  calculateWaitTime(tasks, n); }    其中，两种情况用到的两个函数分别为scheduletasks函数和calculateWaitTime函数。前者的作用是将所有的任务按照CPU占用时间长短进行排序，占用时间短的排在最前面，以保证最终的平均等待时间最短；后者是直接输出任务执行的顺序，并同时将每个任务的开始时间，结束时间，等待时间都输出出来。  void calculateWaitTime(struct task tasks[], int n) {  int i;  int totalwaittime = 0;  for(i = 0; i < n; i++){  if(i==0) tasks[i].starttime = 0;  else tasks[i].starttime = tasks[i-1].finishtime;  tasks[i].finishtime = tasks[i].starttime + tasks[i].servicetime;  tasks[i].waittime = tasks[i].starttime;  totalwaittime += tasks[i].waittime;  printf("任务%d：开始时间：%d，结束时间：%d，等待时间：%d\n", tasks[i].id, tasks[i].starttime, tasks[i].finishtime, tasks[i].waittime);  }  float averagetime = totalwaittime / n;  printf("平均等待时间：%f\n",averagetime);  }  void scheduletasks(struct task tasks[], int n) {  // 排序任务，按照服务时间升序  int i = 0;  int j = 0;  for (i = 0; i < n - 1; i++) {  for (j = 0; j < n - i - 1; j++) {  if (tasks[j].servicetime > tasks[j + 1].servicetime) {  // 交换两个任务的位置  struct task temp = tasks[j];  tasks[j] = tasks[j + 1];  tasks[j + 1] = temp;  }  }  }  }    四、界面设计  ·采用简洁的printf函数实现  ·按提示输入数字即可完成相应的操作    图4-1 —— 界面设计  五、运行测试与分析  1-1.按提交顺序执行，测试数据：  ·任务数：5，  ·每个任务的执行时间：6 9 5 8 4    六、实验收获与思考  1.思考问题：  最短作业优先，存在“长任务饥饿”的问题，即如果动态地不断加入作业，只要提交作业所需要的CPU服务时间比较短，则先提交的长任务将一直得不到服务，如何才能解决该问题？  SJF算法能有效地降低作业的平均等待时间，但也存在一些不容忽视的缺点：  ·长作业进程有可能被饿死。在有短作业(进程)持续不断存在的情况下，由于调度程序总是优先调度那些(即使是后进来的)短作业(进程)，将导致长作业(进程)长期得不到调度而饿死;  ·缺少剥夺机制，不适用于分时系统或交互式事务处理环境。  ·无法准确知道作业进程)的确切执行时间，致使该算法不一定能真正做到短作业优先调度。  解决办法：  可设定一个循环，当循环执行了指定次数时，对后面的为执行的任务进行执行，以此来解决长任务饥饿的问题。  2.实验收获：  经过此次的课程设计，我可以很明确的了解到自己哪些知识点掌握的比较好，哪些知识点运用的还不够熟练。通过此次课程设计也学会综合运用理论知识来分析和解决实际问题的能力。而且我对队列的思想有了更多的理解，对等待时间最短的调度方法有了更加深刻的认识，为以后的深层的学习打下了基础。 |