# 实验一：多级反馈队列调度算法的模拟实现

|  |  |
| --- | --- |
| 实验名称 | 实验一：多级反馈队列调度算法的模拟实现 |
| 姓名学号 | 高丽 2017302207 |
| 1. **实验目的** 2. 用C++模拟实现多级反馈队列调度算法，以更好掌握该调度算法的调度机制。 3. 深入理解多级反馈队列与短进程优先的调度算法、抢占式调度算法之间的差别。 4. 提高使用编程解决实际问题的能力。 | |
| 1. **实验内容** 2. 使用语言：C++ 3. 数据结构：   ① 主要包含两个结构体：process进程结构体、FBqueue多级反馈队列中的队列结构体。   * 进程结构体（process）   struct process {  int No;//进程号  int arrive\_time;//到达时间  int require\_time;//需要执行的时间  int timeslice ;//进程的时间片（这里代表某次调度运行的时间大小）  int haveran\_time;//已经运行的时间  int status;//进程的状态  int current\_time;//进程运行时，当下的时间  };   * 队列结构体（FBqueue）   struct FBqueue {  int timeslice ;//时间片  queue<process>Q;//该级队列  };  ② 定义了两个宏：Ready为0 表示进程就绪， Finished为1表示已完成，均表示进程状态。  ③ 定义FBqueue FBQ[20]表示最大支持20级的队列数组，FBQ[0]为最高级的队列，记作第一级，多级反馈队列的具体级数可按输入设定；定义queue<process> allprocess表示初始时刻所有就绪进程的队列。   1. 流程： 2. 总体思路：（流程图如下）      1. 具体实现：  * 该模拟主要由以下几个函数构成：   bool isarrive();//判断进程在time时刻是否到达  int run\_p();//运行p进程  void printme();//打印进程信息  int whereisme();//返回p在哪一级  void Init();//初始化就绪队列allprocess和FBQ各级队列  void MFQ();//多级反馈队列主体函数   * 算法主体部分为MFQ()函数，MFQ()函数和Init()函数流程已均包含在上面的流程图中。 * run\_p()函数具体说明（流程图如下，包含了whereisme()函数详细说明）：      * isarrive()函数说明：   当p.arrive\_time <= current\_time返回true，否则返回false。 | |
| 1. **实验结果**（红色箭头为输入）   多级反馈队列的级数可以自由设定，最多可达20级，队列时间片与进程数量也可自由设定，进程数量无最大限制。  ① 输入队列级数4，时间片分别为1、2、4、6~~（分别为从低级到高级）~~这句错误，时间片实际上是跟着优先级增加而减少！第一队列优先级最高、时间片最短  ② 输入进程数6。输入6个进程的信息。  结果如下：    该图下方为所有进程信息的输出  ③ 开始模拟： | |
| 1. **结果分析**   输入设定系统有四级队列，最高级为第一级，时间片为6s，第二级4s，第三级2s，第四级1s。  共有6个进程，信息如下：  进程号：2 到达时间：0 需要时间：4  进程号：3 到达时间：1 需要时间：8  进程号：1 到达时间：2 需要时间：12  进程号：4 到达时间：3 需要时间：12  进程号：5 到达时间：4 需要时间：17  进程号：6 到达时间：30 需要时间：10  第0秒。进程[2]到达，处于第一级，时间片为6s，进程运行4s。第4s完成。  第4秒。进程[3]首次运行，处于第一级，运行6s，还需要2s，未完成，放入第二级。  第10秒。进程[1]首次运行，处于第一级，运行6s，还需要3s，未完成，放入第二级。  第16秒。进程[4]首次运行，处于第一级，运行6s，还需要7s，未完成，放入第二级。  第22秒。进程[5]首次运行，处于第一级，运行6s，还需要11s，未完成，放入第二级。  第28秒。无新进程到达，一级队列为空，处于第二级的进程[3]再次运行。  第29秒。进程[6]到达，进入第一级，抢占CPU，进程[3]之前运行了1s，被抢占后放入第二级队尾，运行进程[6]6s，还需4s，未完成，放入第二级。  第35秒。无新进程到达，一级队列为空，处于第二级的进程[1]再次运行，运行3s，完成，第38秒完成。  第38秒。无新进程到达，一级队列为空，处于第二级的进程[4]再次运行，运行4s，还需要3s，未完成，放入第三级队列。  第42秒。无新进程到达，一级队列为空，处于第二级的进程[5]再次运行，运行4s，还需要7s，未完成，放入第三级队列。  第46秒。无新进程到达，一级队列为空，处于第二级的进程[3]再次运行，由于之前运行1s，再运行1s完成，第47秒进程[3]完成。  第47秒。无新进程到达，一级队列为空，处于第二级的进程[6]再次运行，运行4s完成，第51秒进程[6]完成。  第51秒。无新进程到达，一、二级队列为空，处于第三级的进程[4]再次运行，运行2s，还需1s，未完成，放入第四级。  第53秒。无新进程到达，一、二级队列为空，处于第三级的进程[5]再次运行，运行2s，还需5s，未完成，放入第四级。  第55秒。无新进程到达，一、二、三级队列为空，处于第四级的进程[4]再次运行，运行1s完成，第56秒进程[4]完成。  第56秒。无新进程到达，一、二、三级队列为空，处于第四级的进程[5]再次运行1s，还需4s，继续放入最低级。  第57-60秒，均无新进程到达，最低级进程[5]不断运行，每次运行1s，第61秒完成。  第61秒。结束模拟。  以上为对实验结果全部截图的分析，本次调度耗时61s。  本次模拟包含了以下多种情况：  ①某进程的需要运行时间小于第一级时间片，第一级时间片未消耗完就完成。  ②次级有进程运行时，有新进程到达，抢占CPU，次级的进程被打断运行，并放入该级队尾。  ③某进程一二三四级时间片均运行完仍然未完成，继续在最低级按1s时间片运行。  ④各级均有运行。  综上，本次模拟包含了所有情况，实验结果符合算法要求。 | |