|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学　　号： | E12214052 | 专　　业： | 计算机科学与技术 | 姓　名： | 赵宸宇 |
| 实验日期： | 2024年10月18日 | 教师签字： |  | 成　绩： |  |

实验报告

【实验名称】 作业调度

实验项目地址：<https://github.com/YUCHENYUXI/OSLab>

目　录

[一 【实验目的】 3](#_Toc180164787)

[二 【数据结构 和 符号说明】 3](#_Toc180164788)

[2.1 数据结构说明 3](#_Toc180164789)

[2.1.1 结构体 - **进程控制块** PCB: 3](#_Toc180164790)

[2.1.2 优先队列 mylist: 3](#_Toc180164791)

[2.1.3 优先队列 sjf\_queue 3](#_Toc180164792)

[2.2 符号说明表 3](#_Toc180164793)

[三 【流程图（算法流程图）概览】 4](#_Toc180164794)

[四 【源程序（须有注释,含学号、姓名）】 5](#_Toc180164795)

[五 【运行时的初值和结果概览（结果中须显示学号、姓名）】 12](#_Toc180164796)

[5.1 FCFS用例1： 12](#_Toc180164797)

[5.1.1 理论结果： 12](#_Toc180164798)

[5.1.2 运行结果： 12](#_Toc180164799)

[5.1.3 校验方法及结果： 13](#_Toc180164800)

[5.2 FCFS用例2： 13](#_Toc180164801)

[5.2.4 理论结果： 13](#_Toc180164802)

[5.2.5 运行结果： 14](#_Toc180164803)

[5.2.6 校验方法及结果： 15](#_Toc180164804)

[5.3 SJF用例（同FCFS用例2）： 15](#_Toc180164805)

[5.3.7 理论结果： 15](#_Toc180164806)

[5.3.8 运行结果： 15](#_Toc180164807)

[5.3.9 校验方法及结果： 16](#_Toc180164808)

[六 【实验内容】 16](#_Toc180164809)

[七 【实验原理】 17](#_Toc180164810)

[八 【实验步骤】 17](#_Toc180164811)

[8.1 FCFS算法实现 17](#_Toc180164812)

[8.1.1 准备工作 17](#_Toc180164813)

[8.1.2 算法实现 18](#_Toc180164814)

[8.1.3 算法描述 21](#_Toc180164815)

[8.1.4 代码逻辑概述 21](#_Toc180164816)

[8.1.5 算法测试 21](#_Toc180164817)

[8.2 SJF算法 22](#_Toc180164818)

[8.2.6 特点： 22](#_Toc180164819)

[8.2.7 算法逻辑 22](#_Toc180164820)

[8.2.8 算法实现 23](#_Toc180164821)

[8.2.9 算法测试 27](#_Toc180164822)

[九 【小结与讨论】 27](#_Toc180164823)

[9.1 小结： 27](#_Toc180164824)

[9.2 FCFS 27](#_Toc180164825)

[9.3 SJF 27](#_Toc180164826)

[9.4 讨论 28](#_Toc180164827)

# 【实验目的】

巩固和加深作业调度的概念。

# 【数据结构 和 符号说明】

## 数据结构说明

* + 1. 结构体 - **进程控制块** PCB:

**描述**：PCB 代表进程控制块（Process Control Block），用于存储进程的基本信息。

**成员**：

std::string name: 进程名称。

int arrive: 进程到达时间。

int exetime: 进程执行时间。

**运算符重载**：bool operator<(const PCB& other) const，该运算符重载用于在优先队列中比较 PCB 对象。

在FCFS中以到达时间最早进行排序（到达时间较小的优先级高）。

在SJF中以运行时间最短进行排序（到达时间较小的优先级高）。

**用途**：构建ps（PCB[]），用来初始化进程数组，包含所有待调度的进程的PCB。

* + 1. 优先队列 mylist:

类：std::priority\_queue<PCB, std::vector<PCB>>

描述：用于存储和管理 PCB 对象的队列，按照到达时间优先排序。

支撑数据结构：使用 std::vector<PCB> 作为底层容器。

* + 1. 优先队列 sjf\_queue

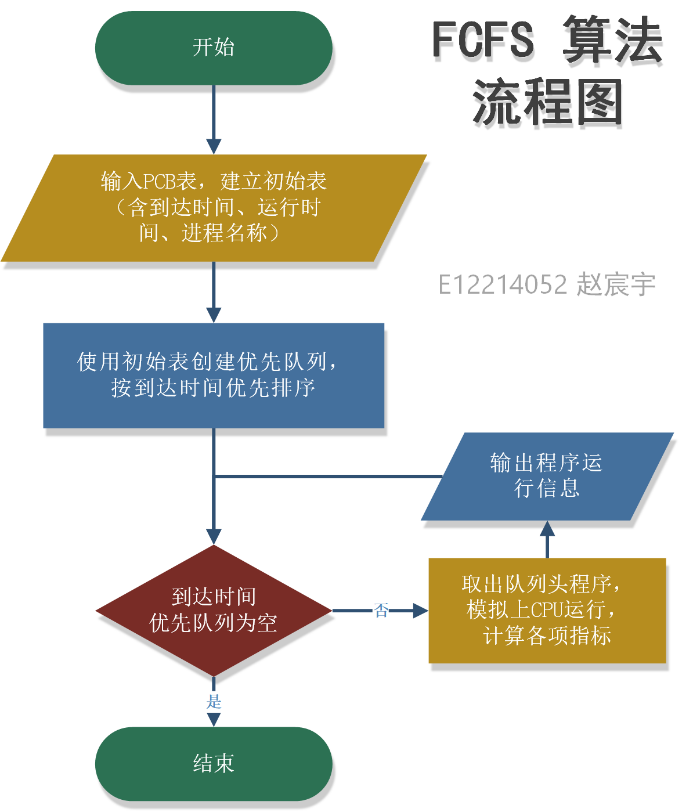
类：std::priority\_queue<PCB, std::vector<PCB>>

SJF队列，基于优先队列算法以实现短作业优先调度。

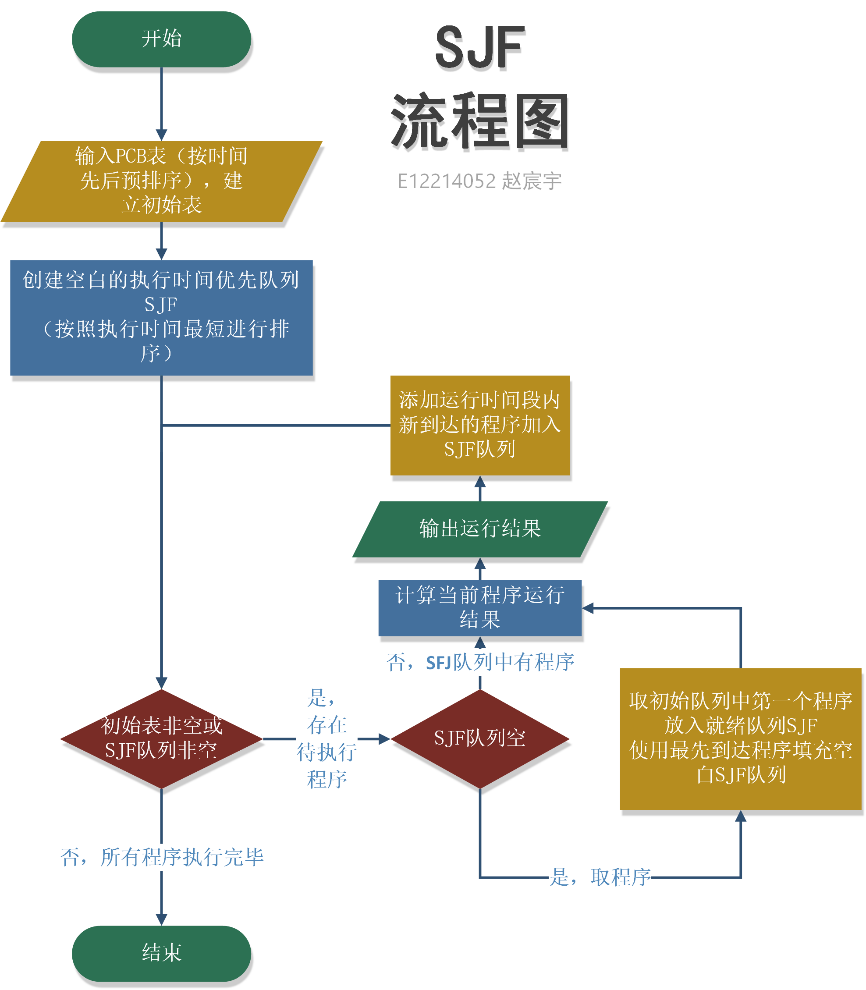
## 符号说明表

|  |  |
| --- | --- |
| 符号 | 说明 |
|  | 进程名称 |
|  | 进程到达时间 |
|  | 进程执行时间 |
|  | 按到达时间排序的优先队列 |
|  | 累计运行时间 |
|  | 进程标识符，从1开始计算 |
|  | 当前正在执行的进程的映像 |
|  | 周转时间（完成时间 - 到达时间） |
|  | 带权周转时间（TAR / 执行时间） |
|  | SJF队列，使用优先队列以实现短作业优先调度。 |
|  | 上次累计运行时间，用于跟踪进程执行前的时间。 |

# 【流程图（算法流程图）概览】



Figure



Figure

# 【源程序（须有注释,含学号、姓名）】

表 1

|  |
| --- |
| FCFS程序 |
| #include<iostream>  #include<algorithm>  #include<queue>  #include <vector>  #include<sstream>  // using namespace std;  // E12214052 赵宸宇  struct PCB { // def process control block      std::string name;      int arrive;      int exetime;      bool operator<(const PCB& other) const {          return arrive > other.arrive ;          // 先按到达时间排序，后按照执行时间排序      }  };  /\*  先来先服务(FCFS)调度算法  　FCFS是最简单的调度算法，该算法既可用于作业调度，也可用于进程调度。  　当在作业调度中采用该算法时，系统将按照作业到达的先后次序来进行调度，或者说它是优先考虑在系统中等待时间最长的作业，而不管该作业所需执行时间的长短，从后备作业队列中选择几个最先进入该队列的作业，将它们调入内存，为它们分配资源和创建进程，然后把它放入就绪队列。  \*/  /\*  \* self:          {"QQ",12,0},          {"Wechat",13,0},          {"Vscode",5,4},          {"Pycharm",9,2},          {"Python-GPU0",39,6},          {"Chrome",3,3},          {"System",4,9}  \* demo 2:          {"QQ",0,4},          {"Wechat",1,3},          {"Vscode",2,5},          {"Pycharm",3,2},          {"Python-GPU0",4,4}  \* demo 1:          {"QQ",0,1},          {"Wechat",1,100},          {"Vscode",2,1},          {"Pycharm",3,100}    \*/  int main() {      std::cout << "初始化PCB列表\n" << std::endl;      PCB ps[] = {          {"QQ",0,1},          {"Wechat",1,100},          {"Vscode",2,1},          {"Pycharm",3,100}      };      // 建立到达时间优先队列      std::priority\_queue<PCB, std::vector<PCB>> mylist;      // 逐个导入PCB到表中      for (auto pcb : ps) {          mylist.push(pcb);      }      // 运行      int accTime = 0;      int PID = 0;      std::cout << "PID 从1开始计算：\n";      while (!mylist.empty()) {          // 模拟 就绪队列 上CPU          PID++;          std::cout <<"[PID]:"<< PID << std::endl;          PCB current = mylist.top();          mylist.pop();          // 模拟运行          std::cout << "[sys]   : "<< current.name <<"上CPU " << std::endl;          std::cout << "到达时间: " << current.arrive << std::endl;          std::cout << "服务时间: " << current.exetime << std::endl;          // 完成时间          accTime += current.exetime;          std::cout << "完成时间: " << accTime << std::endl;          // 周转时间          int TAR = accTime - current.arrive;          std::cout << "周转时间: " << TAR << std::endl;          // 带权周转时间          float wTAR = TAR\*1.0f/current.exetime;          std::cout << "带权TAR : " << wTAR << std::endl << std::endl;        }        return 0;  }  // E12214052 赵宸宇 |

表 2

|  |
| --- |
| SJF程序 |
| #include<iostream>  #include<algorithm>  #include<queue>  #include<vector>  #include<sstream>  // E12214052赵宸宇  struct PCB { // def process control block      std::string name;      int arrive;      int exetime;      bool operator<(const PCB& other) const {          return exetime > other.exetime;          // 按照执行时间排序，执行时间越短优先级越高      }  };  /\*  短作业优先的调度算法   SJF算法是以作业的长短来计算优先级，作业越短，其优先级越高。   作业的长短是以作业所要求的运行时间来衡量的。SJF算法可以分别用于作业调度和进程调度。   在把短作业优先调度算法用于作业调度时，   它将从外存的作业后备队列中选择若干个估计运行时间最短的作业，   优先将它们调入内存运行。  \*/  /\*  \* self:          {"QQ",12,0},          {"Wechat",13,0},          {"Vscode",5,4},          {"Pycharm",9,2},          {"Python-GPU0",39,6},          {"Chrome",3,3},          {"System",4,9}  \* demo 2:          {"QQ",0,4},          {"Wechat",1,3},          {"Vscode",2,5},          {"Pycharm",3,2},          {"Python-GPU0",4,4}  \* demo 1:          {"QQ",0,1},          {"Wechat",1,100},          {"Vscode",2,1},          {"Pycharm",3,100}  \*/  int main() {      std::cout << "初始化PCB列表\n" << std::endl;      PCB ps[] = {          {"A",0,4},          {"B",1,3},          {"C",2,5},          {"D",3,2},          {"E",4,4}      };      // 建立到达时间优先队列      std::vector<PCB> arrival\_list(std::begin(ps), std::end(ps));      // 建立空的SJF队列      std::priority\_queue<PCB, std::vector<PCB>> sjf\_queue;      // 当前累计时间      int accTime = 0,preacc=0 ;      int PID = 0;      std::cout << "PID 从1开始计算：\n";      while (!arrival\_list.empty() || !sjf\_queue.empty()) {          // 当SJF队列为空时，从非空的到达队列中取出一个进程          if (sjf\_queue.empty() && !arrival\_list.empty()) {              sjf\_queue.push(\*arrival\_list.begin());              arrival\_list.erase(arrival\_list.begin());          }          // 运行SJF队列中最短的作业          PCB current = sjf\_queue.top();          sjf\_queue.pop();          PID++;          preacc = accTime;          accTime += current.exetime;          std::cout << "[PID]:" << PID << std::endl;          std::cout << "[sys]   : " << current.name << "上CPU " << std::endl;          std::cout << "到达时间: " << current.arrive << std::endl;          std::cout << "服务时间: " << current.exetime << std::endl;          // 模拟运行          std::cout << "完成时间: " << accTime << std::endl;          // 周转时间          int TAR = accTime - current.arrive;          std::cout << "周转时间: " << TAR << std::endl;          // 带权周转时间          float wTAR = TAR\*1.0f / current.exetime;          std::cout << "带权TAR : " << wTAR << std::endl << std::endl;          bool have = 1;          while (have && !arrival\_list.empty()) {              auto it = \*arrival\_list.begin();              if (it.arrive < accTime) {                  have = true;                  sjf\_queue.push(\*arrival\_list.begin());                  arrival\_list.erase(arrival\_list.begin());              }              else {                  have = false;              }          }      }  // E12214052赵宸宇      return 0;  } |

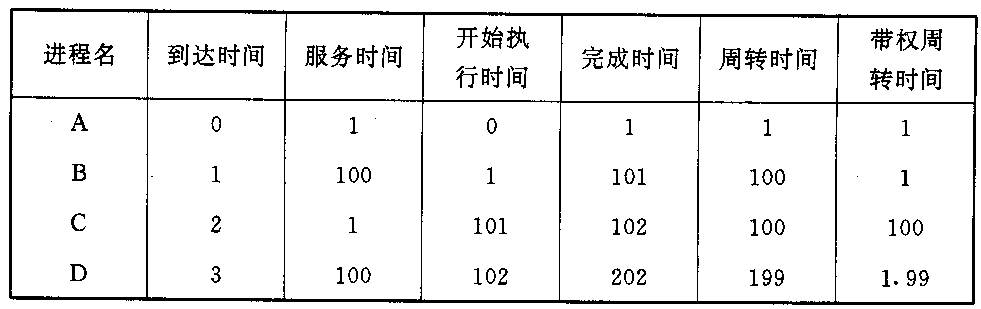
# 【运行时的初值和结果概览（结果中须显示学号、姓名）】

本实验对PPT中的结果进行了模拟，得到了正确的结果。同时**还发现了PPT中的一处错误**：即SJF调度算法之**C程序的带权周转时间计算有误**。

用例和测试结果如下所示：

## FCFS用例1：

* + 1. 理论结果：



Figure

* + 1. 运行结果：

文本

描述已自动生成

Figure

图片包含 表格

描述已自动生成

Figure

* + 1. 校验方法及结果：

**目视检查，一致；**

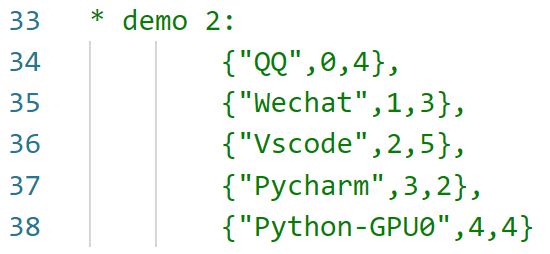
## FCFS用例2：

* + 1. 理论结果：



Figure

* + 1. 运行结果：



Figure



Figure

文本

低可信度描述已自动生成

Figure

* + 1. 校验方法及结果：

**目视检查，一致；**

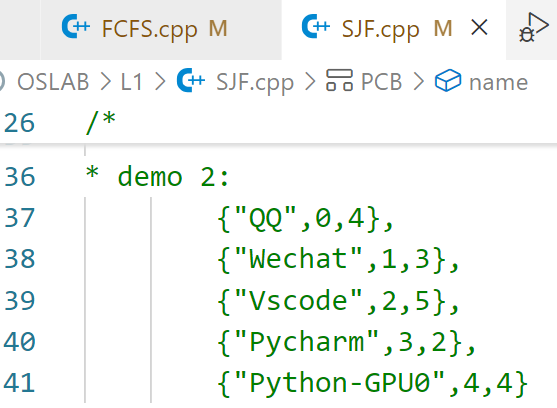
## SJF用例（同FCFS用例2）：

* + 1. 理论结果：



Figure

* + 1. 运行结果：



Figure

一些文字和图案

描述已自动生成

Figure

* + 1. 校验方法及结果：

目视检查，**发现PPT有一处（3.1）计算错误**。实验结果正确；

# 【实验内容】

设计调度算法，模拟实现作业调度。

设计先来先服务 FCFS调度算法；设计按短作业优先 SJF调度算法

# 【实验原理】

先来先服务(FCFS)调度算法：  
　　FCFS是最简单的调度算法，该算法既可用于作业调度，也可用于进程调度。  
 当在作业调度中采用该算法时，系统将按照作业到达的先后次序来进行调度，或者说它是优先考虑在系统中等待时间最长的作业，而不管该作业所需执行时间的长短，从后备作业队列中选择几个最先进入该队列的作业，将它们调入内存，为它们分配资源和创建进程，然后把它放入就绪队列。

短作业优先的调度算法：  
　　SJF算法是以作业的长短来计算优先级，作业越短，其优先级越高。作业的长短是以作业所要求的运行时间来衡量的。SJF算法可以分别用于作业调度和进程调度。在把短作业优先调度算法用于作业调度时，它将从外存的作业后备队列中选择若干个估计运行时间最短的作业，优先将它们调入内存运行。

实验参考：

1. 1操作系统实验\_\_实验一 作业调度
2. Program for FCFS Scheduling|Geeks4Geeks <https://www.geeksforgeeks.org/program-for-fcfs-cpu-scheduling-set-1/>
3. CPU Scheduling in Operating Systems|Geeks4Geekshttps://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/std::priority\_queue <https://en.cppreference.com/w/cpp/container/priority_queue>

# 【实验步骤】

## FCFS算法实现

* + 1. 准备工作

通过复习书本、看PPT、看GeeksForGeeks等的介绍。首先建立对两大算法的基本印象，然后思考实现细节。

FCFS是指一种排队系统，其中第一个到达的程序将首先被处理，FCFS也即FIFO，可以用队列实现。

作为一种非抢占式CPU调度算法（CPU Scheduling），FCFS规定，最先请求 CPU 的进程将首先获得 CPU。当进程进入就绪队列时，其PCB链接到队列尾部。当 CPU 空闲时，将 CPU 分配给队列头部的进程。然后，正在运行的进程将从队列中移除。

FCFS的优点是：先到先得，确保程序总会在某个时刻被执行，不会导致饥饿。易于实现，且实现该算法的额外开销很小，不会对CPU性能造成影响。

FCFS（非抢占式）的缺点是：对紧急的任务不能及时响应，这样一来，就不能适应实时系统的时间需求；没有考虑响应比、等待时间、运行时间，没有规定单个程序对CPU占用的时间限制，整体效率低。

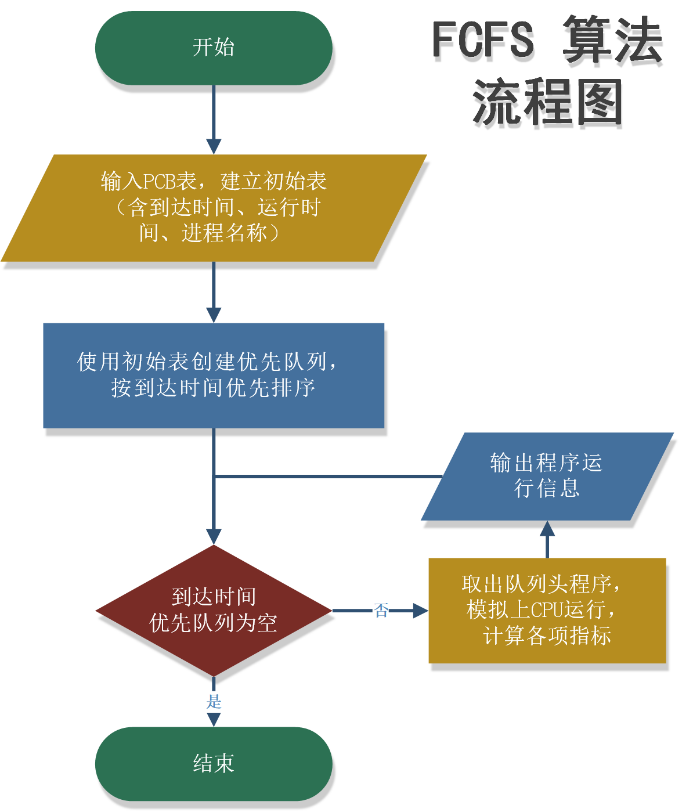
* + 1. 算法实现

基于对FCFS算法的认识，以下程序被编写出来以实现这个算法：

表 1

|  |
| --- |
| FCFS程序 |
| #include<iostream>  #include<algorithm>  #include<queue>  #include <vector>  #include<sstream>  // using namespace std;  // E12214052 赵宸宇  struct PCB { // def process control block      std::string name;      int arrive;      int exetime;      bool operator<(const PCB& other) const {          return arrive > other.arrive ;          // 先按到达时间排序，后按照执行时间排序      }  };  int main() {      std::cout << "初始化PCB列表\n" << std::endl;      PCB ps[] = {          {"QQ",0,1},          {"Wechat",1,100},          {"Vscode",2,1},          {"Pycharm",3,100}      };      // 建立到达时间优先队列      std::priority\_queue<PCB, std::vector<PCB>> mylist;      // 逐个导入PCB到表中      for (auto pcb : ps) {          mylist.push(pcb);      }      // 运行      int accTime = 0;      int PID = 0;      std::cout << "PID 从1开始计算：\n";      while (!mylist.empty()) {          // 模拟 就绪队列 上CPU          PID++;          std::cout <<"[PID]:"<< PID << std::endl;          PCB current = mylist.top();          mylist.pop();          // 模拟运行          std::cout << "[sys]   : "<< current.name <<"上CPU " << std::endl;          std::cout << "到达时间: " << current.arrive << std::endl;          std::cout << "服务时间: " << current.exetime << std::endl;          // 完成时间          accTime += current.exetime;          std::cout << "完成时间: " << accTime << std::endl;          // 周转时间          int TAR = accTime - current.arrive;          std::cout << "周转时间: " << TAR << std::endl;          // 带权周转时间          float wTAR = TAR\*1.0f/current.exetime;          std::cout << "带权TAR : " << wTAR << std::endl << std::endl;        }        return 0;  }  // E12214052 赵宸宇 |

* + 1. 算法描述



Figure

如上图所示，是FCFS算法的流程描述。在该算法中，第一个进程的等待时间为 0，所以它最先执行。后面的程序依照先来后到的队列顺序执行。

* + 1. 代码逻辑概述

1. 定义 PCB 结构体以存储每个进程的基本信息。
2. 初始化一个 PCB 数组，包含若干个进程。
3. 使用优先队列 mylist 将 PCB 数组中的每个进程按照到达时间的优先级压入。
4. 模拟进程调度过程：从优先队列中弹出进程，计算并输出其完成时间、周转时间和带权周转时间。
5. 重复步骤4，直到所有进程都被调度完毕。
   * 1. 算法测试

见上文——五【运行时的初值和结果概览（结果中须显示学号、姓名）】。

## SJF算法

**Shortest job next** (**SJN**), a.k.a. **shortest job first** (**SJF**) a.k.a. **shortest process next** (**SPN**)。

SJF作为一种调度策略，总是选择已经到达的进程中执行时间最短的进程进行执行。

* + 1. 特点：

SJF 是一种非抢占式算法。它最小化了每个进程在执行完成前等待的平均时间，**在所有调度算法中，它的平均等待时间最短**。

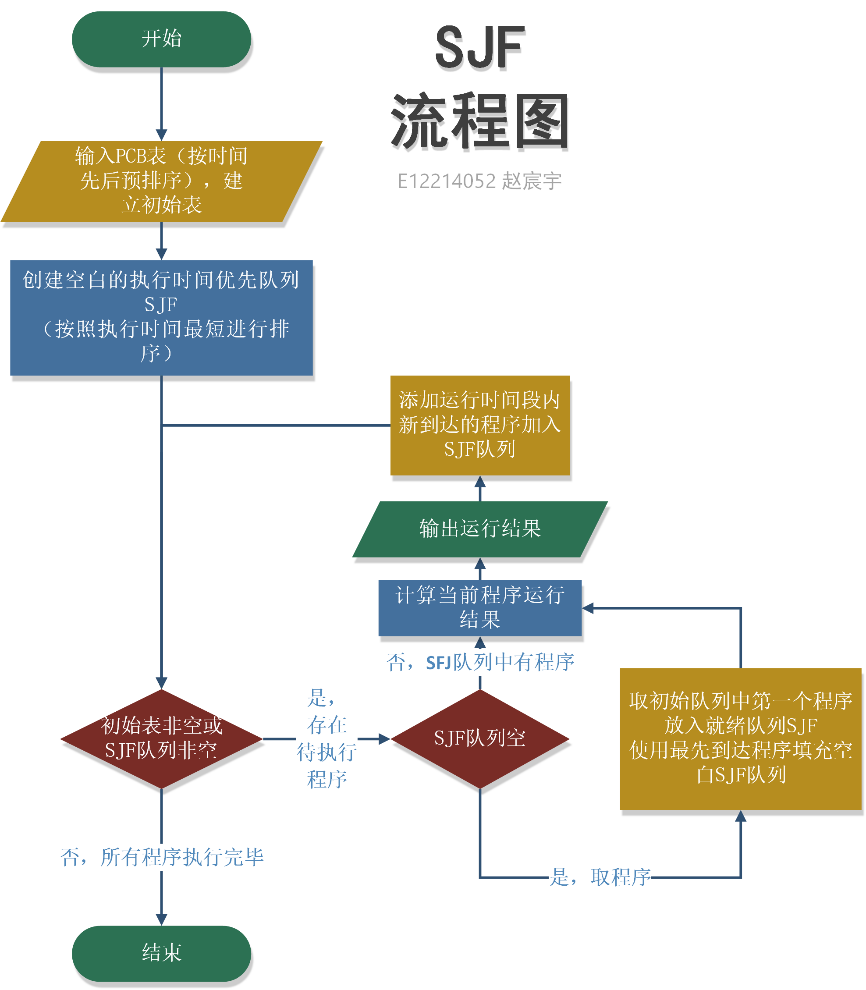
是一个贪婪算法。

较短的过程持续进行，可能会导致饥饿。这个问题可以用ageing算法来解决

在执行之前必须知道作业的总执行时间，在实践中是不可行的，因为操作系统可能不知道突发时间，因此可能无法对其进行排序。

虽然无法预测执行时间，但可以使用多种方法来估计作业的执行时间，例如先前执行时间的加权平均值。

* + 1. 算法逻辑



Figure

按照到达时间对所有流程进行排序。

然后选择具有最早到达时间和最短运行时间的进程。

该进程完成后，创建一个在该过程完成之前到达的所有进程的进程池，然后从池中选择具有最短运行时间的进程执行，并重复这个步骤直到系统关机。

* + 1. 算法实现

表 2

|  |
| --- |
| SJF程序 |
| #include<iostream>  #include<algorithm>  #include<queue>  #include<vector>  #include<sstream>  // E12214052赵宸宇  struct PCB { // def process control block      std::string name;      int arrive;      int exetime;      bool operator<(const PCB& other) const {          return exetime > other.exetime;          // 按照执行时间排序，执行时间越短优先级越高      }  };  /\*  短作业优先的调度算法   SJF算法是以作业的长短来计算优先级，作业越短，其优先级越高。   作业的长短是以作业所要求的运行时间来衡量的。SJF算法可以分别用于作业调度和进程调度。   在把短作业优先调度算法用于作业调度时，   它将从外存的作业后备队列中选择若干个估计运行时间最短的作业，   优先将它们调入内存运行。  \*/  /\*  \* self:          {"QQ",12,0},          {"Wechat",13,0},          {"Vscode",5,4},          {"Pycharm",9,2},          {"Python-GPU0",39,6},          {"Chrome",3,3},          {"System",4,9}  \* demo 2:          {"QQ",0,4},          {"Wechat",1,3},          {"Vscode",2,5},          {"Pycharm",3,2},          {"Python-GPU0",4,4}  \* demo 1:          {"QQ",0,1},          {"Wechat",1,100},          {"Vscode",2,1},          {"Pycharm",3,100}  \*/  int main() {      std::cout << "初始化PCB列表\n" << std::endl;      PCB ps[] = {          {"A",0,4},          {"B",1,3},          {"C",2,5},          {"D",3,2},          {"E",4,4}      };      // 建立到达时间优先队列      std::vector<PCB> arrival\_list(std::begin(ps), std::end(ps));      // 建立空的SJF队列      std::priority\_queue<PCB, std::vector<PCB>> sjf\_queue;      // 当前累计时间      int accTime = 0,preacc=0 ;      int PID = 0;      std::cout << "PID 从1开始计算：\n";      while (!arrival\_list.empty() || !sjf\_queue.empty()) {          // 当SJF队列为空时，从非空的到达队列中取出一个进程          if (sjf\_queue.empty() && !arrival\_list.empty()) {              sjf\_queue.push(\*arrival\_list.begin());              arrival\_list.erase(arrival\_list.begin());          }          // 运行SJF队列中最短的作业          PCB current = sjf\_queue.top();          sjf\_queue.pop();          PID++;          preacc = accTime;          accTime += current.exetime;          std::cout << "[PID]:" << PID << std::endl;          std::cout << "[sys]   : " << current.name << "上CPU " << std::endl;          std::cout << "到达时间: " << current.arrive << std::endl;          std::cout << "服务时间: " << current.exetime << std::endl;          // 模拟运行          std::cout << "完成时间: " << accTime << std::endl;          // 周转时间          int TAR = accTime - current.arrive;          std::cout << "周转时间: " << TAR << std::endl;          // 带权周转时间          float wTAR = TAR\*1.0f / current.exetime;          std::cout << "带权TAR : " << wTAR << std::endl << std::endl;          bool have = 1;          while (have && !arrival\_list.empty()) {              auto it = \*arrival\_list.begin();              if (it.arrive < accTime) {                  have = true;                  sjf\_queue.push(\*arrival\_list.begin());                  arrival\_list.erase(arrival\_list.begin());              }              else {                  have = false;              }          }      }  // E12214052赵宸宇      return 0;  } |

* + 1. 算法测试

请见小节—— 5.3 SJF用例（同FCFS用例2）。

# 【小结与讨论】

## 小结：

经过本次实验，我对FCFS、SJF算法的理解更深，对c++程序调试经验更丰富。

通过本次实验我学习到：

## FCFS

FCFS是先来先服务的排队系统,等同于FIFO,可用队列实现。作为非抢占式CPU调度算法,最先请求CPU的进程先获得CPU。新进程加入队尾,CPU空闲时分配给队首进程,运行中进程移出队列。

优点:

公平,避免饥饿

易实现,开销小

缺点:

不适合实时系统,紧急任务响应慢

忽视响应比、等待和运行时间

无CPU占用时限,效率低

## SJF

SJF(最短作业优先)调度策略:

选择已到达且执行时间最短的进程

特点：

非抢占式,平均等待时间最短

贪婪算法,可能导致饥饿(可用ageing解决)

需预知执行时间,实践中难实现

可用加权平均等方法估算执行时间

## 讨论

该实验虽然已经实现，但仍存在不足。

如：

SJF算法中preacc变量并未被使用，但preacc变量应当被使用或被删去以提供更多功能或节省程序空间。

FCFS和SJF的算法只是实现了算法功能，而在算法性能上不够强劲，算法代码也不够精简，对比真实OS程序，可能存在性能上的不足，和Robust不足。

为此，在之后的学习中，我应当考虑向开源的OS代码学习，以提升OS编写能力。

赵宸宇

2024年10月18日