**模拟进程调度与算法分析**

1. **小组成员与分工：**

|  |  |  |
| --- | --- | --- |
| 学号 | 姓名 | 分工 |
| 12348102 | 潘海智 | 组长。负责项目架构的设计与约定；具体实现6种调度算法；文档书写；展示 |
| 12348106 | 邱恒 | 查找linux与windows调度算法知识；提供基本调度算法的伪代码 |
| 12348104 | 潘职林 | 查找实验前所确定难点的相关资料，并提交解决方案 |
| 12348128 | 伍文蕾 | 展示PPT设计与制作；书写文档 |
| 12348137 | 许楚萍 | 负责项目的测试，为所有算法设计样例并分析；书写算法分析文档；展示 |

1. **项目主页：**

<https://github.com/panhzh3/os_homework>

**三.实现工具：**



平台：Linux



语言：python

测试：shell script

具体平台与工具为：

|  |  |
| --- | --- |
| Linux | Linux 3.11.0-23-generic #40~precise1-Ubuntu SMP Wed Jun 4 22:07:06 UTC 2014 i686 i686 i386 GNU/Linux |
| Python | Python2.7 |
| shell | zsh |

**四.架构：**

（蓝色 表示文件夹，红色 表示可执行文件，**加粗** 表示项目关键文件）

os / ─**cpu.py** //模拟cpu，具有时间分片中断功能，基本寄存器

│─**ram.py** //模拟内存，有PCB表、进程队列等

│─**monitor.py** //模拟显示器，实时显示进程调度、cpu状态、统计数据

│─**kernel.py** //模拟内核，处理中断，传递信号，调用调度函数

│─README.md //github上项目的主页文档

│─**pro/** //pro文件夹是进程文件夹，将进程文件拖进即可加入进程

│─tmp/ //tmp文件夹是临时文件夹，用于放临时进程

│─**run** //快速展示效果，使用shell script实现，运行方法见下文

│─**start** //start是本项目主程序，运行方法见下文

│─pre/ //展示文档、PPT、设计进程数据

│─show/ //测试进程文件夹，其中进程文件用于run脚本，不可删

│─pic/ //github上项目主页的展示图片

└─**algo/**─**fcfs.py** //algo文件夹下是具体调度算法，有：先来先服务

│─**rr.py** //轮转，可设置时间片

│─**spn.py** //最短进程优先

│─**srt.py** //最短剩余优先

│─**hrrn.py** //最高响应比优先

└─**fb.py** //反馈，可设置时间片

**五.运行方法：**

1.可执行： ./run 快速观看各种调度算法的展示效果

2.可执行： ./start [algoname,[slicetime]] （algoname必选，slicetime可选）

然后将已经写好的进程文件(tmp目录下)拷贝进pro文件夹下（或者鼠标 拖拽），即相当于手动执行长程调度，加入新进程。

例如:

./start fcfs

./start rr 1

./start rr 10

./start spn

./start srt

./start fb 2

./start hrrn

（注意：因为是模拟，为了展示效果，设置每条进程指令执行时间为1秒，操作系统调度时间为0.5秒，显示器刷新周期为0.15秒）

1. **调度算法分析**

【FCFS】

**算法原理**：

当每个进程就绪后，它就加入就绪队列。当前正在运行的进程停止执行时， 选择在就绪队列中存在时间最长的进程运行。

**优劣势**：

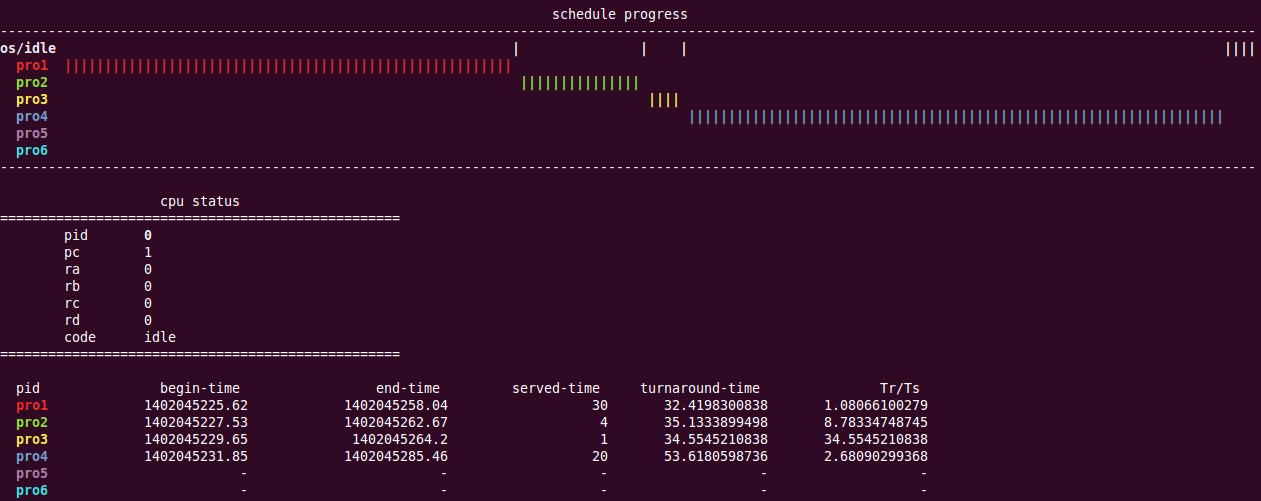
1. 执行长进程比执行短进程更好

2. 相对于I/O密集型的进程，FCFS更有利于处理器密集型的进程

**分析**：

一般I/O密集型进程所需要的处理器时间较短（所以I/O密集型进程应该 优先服务，这样才能更好地利用外设）。从CPU角度，I/O密集型进程相当 于短进程，处理器密集型进程相当于短进程。

**示例：**



【RR】

**基本原理：**

系统将所有就绪进程按FCFS的原则，排成一个队列，依次调度，把CPU 分配给队首进程，并令其执行一个时间片/CPU时间，通常为几个毫秒～几 百毫秒。时间片用完后，该进程将被抢占并插入就绪队列末尾.

**优点：**

在时间片长度设置合理的情况下，更有利于短进程的调度。

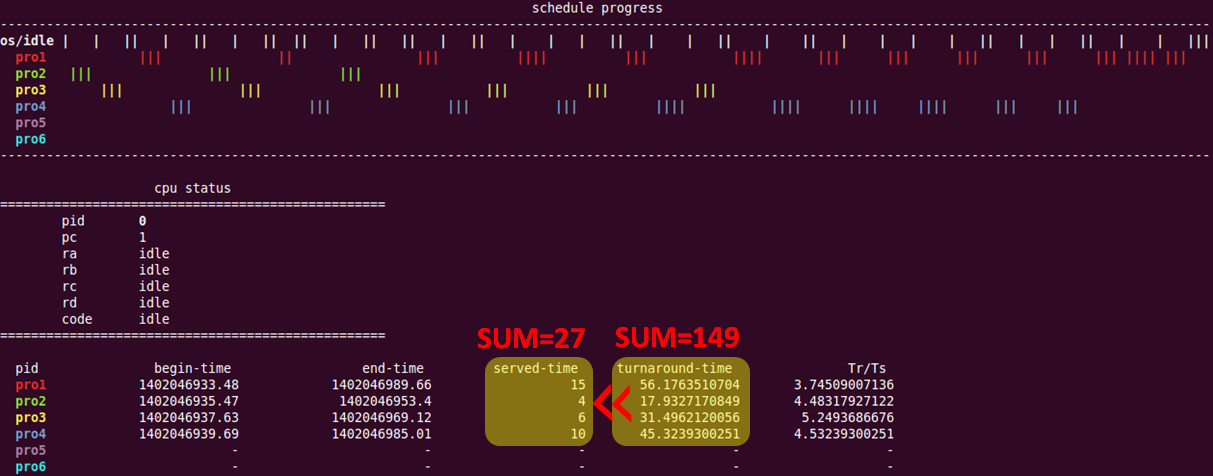
**分析：**

对于轮转法，最主要的设计问题是时间片的长度。由于处理时钟中断、执 行调度和分派函数都需要处理器开销。若时间片太短，频繁的切换进程会 造成处理器开销大；若时间片太长，轮换法就退化成FCFS。

**示例：**

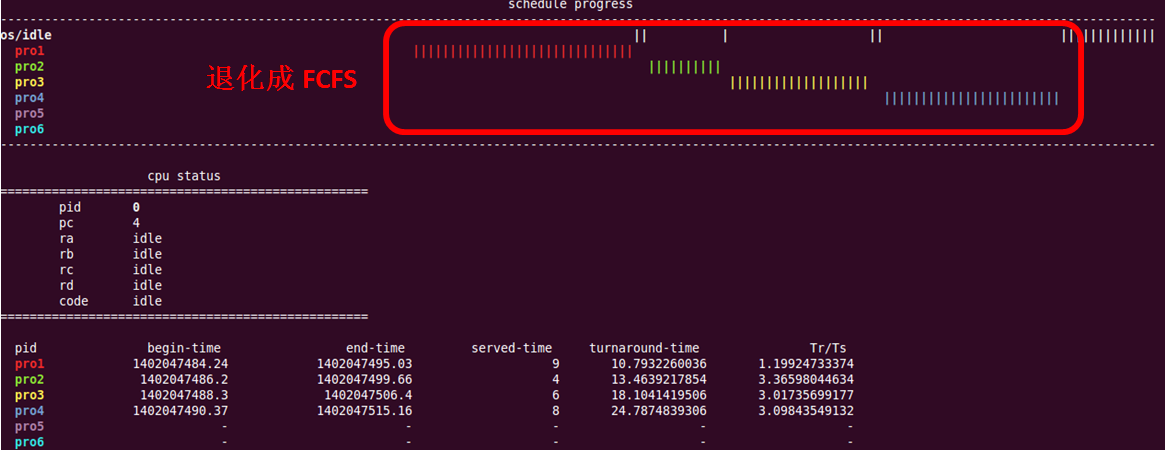
1. 时间片太短（q=1）

|  |  |  |
| --- | --- | --- |
|  | 到达时间 | 服务时间 |
| pro1 | 0 | 15 |
| pro2 | 2 | 4 |
| pro3 | 4 | 6 |
| pro4 | 6 | 10 |



1. 时间片太长（q=10）

|  |  |  |  |
| --- | --- | --- | --- |
|  | 到达时间 | 服务时间 | 备注 |
| pro1 | 0 | 15 |  |
| pro2 | 2 | 4 | 长进程 |
| pro3 | 4 | 6 |  |
| pro4 | 6 | 10 |  |



【SPN】

**基本思想：**

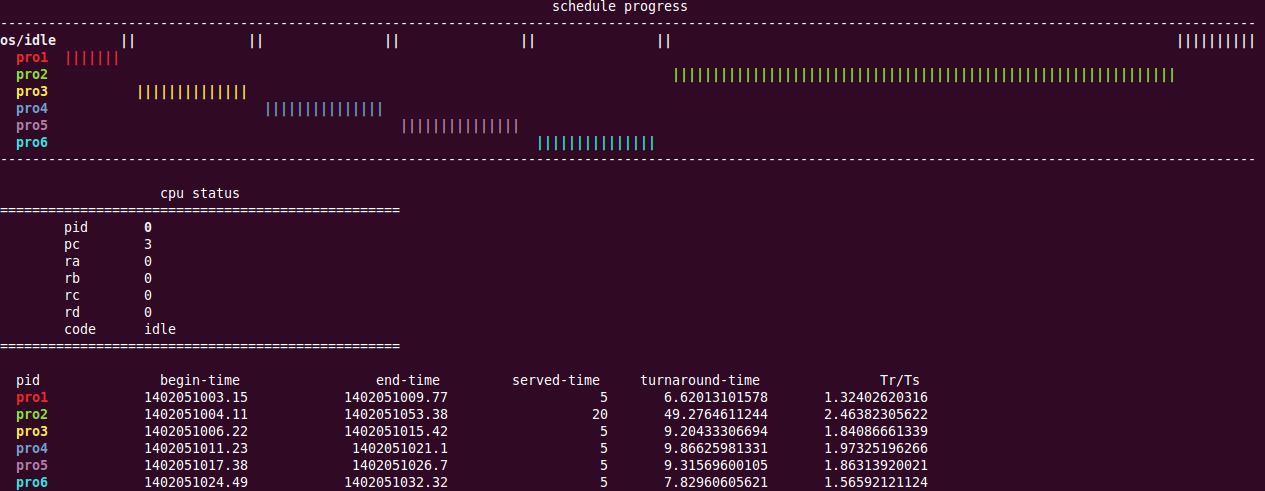
从就绪队列中选出一估计运行时间最短的进程，将处理器分配给它。

**优缺点：**

1. 降低作业的平均等待时间，提高系统吞吐量。
2. 对长作业不利；未考虑作业的紧迫程度。
3. 对进程估计执行时间难以预测。

**示例：**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 到达时间 | 服务时间 | 备注 |
| pro1 | 0 | 5 |  |
| pro2 | 1 | 20 | 长进程Tr/Ts大 |
| pro3 | 3 | 5 |  |
| pro4 | 8 | 5 |  |
| pro5 | 14 | 5 |  |
| pro6 | 21 | 5 |  |



【SRT】

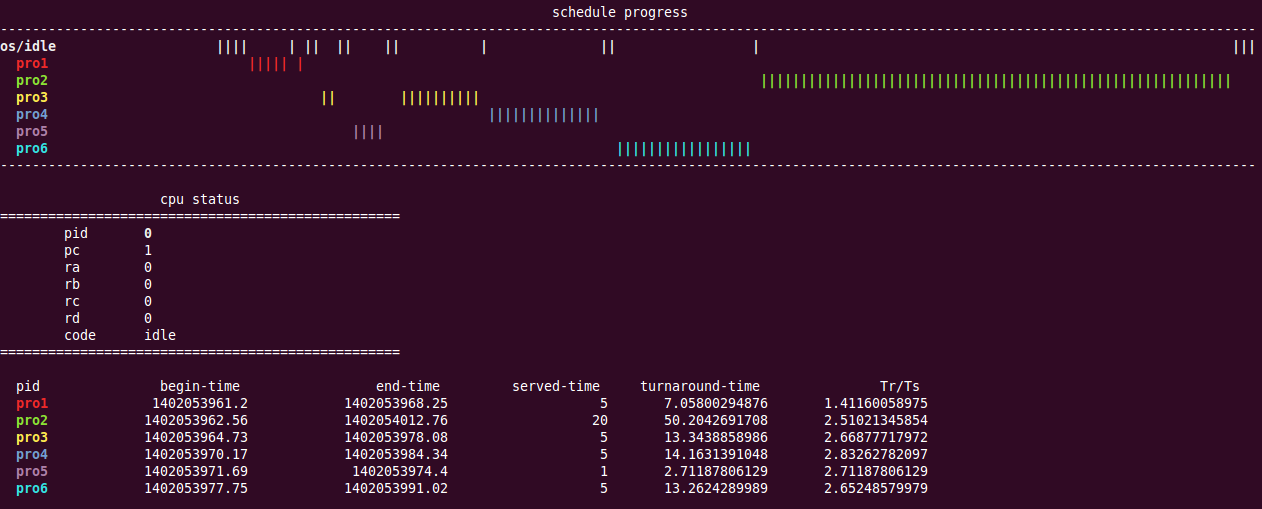
**基本原理：**

SRT是针对SPN增加了抢占机制的版本。

**示例：**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 到达时间 | 服务时间 | 备注 |
| pro1 | 0 | 5 |  |
| pro2 | 1 | 20 |  |
| pro3 | 3 | 5 | 被抢断进程 |
| pro4 | 8 | 5 |  |
| pro5 | 9 | 1 | 抢断进程 |
| pro6 | 15 | 5 |  |

**pro5抢占处理器**



【HRRN】

**基本原理：**

优先权=（等待时间+要求服务时间）/要求服务时间

1. 如果进程的等待时间相同，则要求服务的时间愈短，其优先权愈高，因而

算法有利于短作业；  
（2）当要求服务的时间相同时，作业的优先权决定于其等待时间，因而实现了先来先服务；  
（3）对于长作业，当其等待时间足够长时，其优先权便可升到很高，从而也可获得处理机。

**分析：**

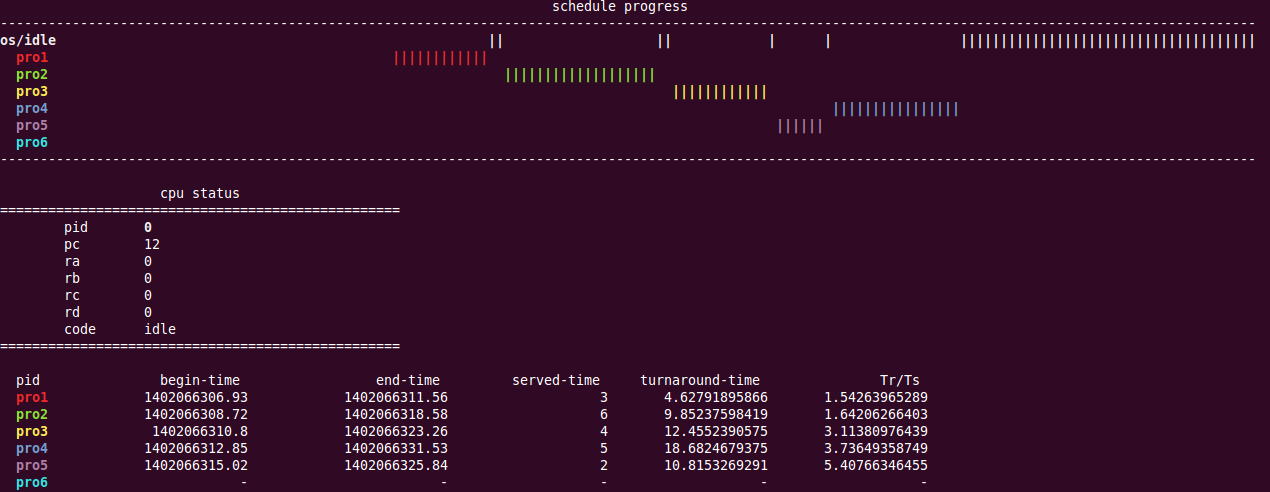
该算法既照顾了短进程，又考虑了进程到达的先后顺序，也不会使进程长期得不

服务。因此，该算法实现了一种较好的折衷。当然，再利用该算法时，每要进行

度之前，都需先进行响应比的计算，这会增加系统的开销。

**示例：**

|  |  |  |
| --- | --- | --- |
|  | 到达时间 | 服务时间 |
| pro1 | 0 | 3 |
| pro2 | 2 | 6 |
| pro3 | 4 | 4 |
| pro4 | 6 | 5 |
| pro5 | 8 | 2 |



【FB】

**基本原理：**

调度基于抢占原则（按时间片）并使用动态优先级机制。新加入的进程放在RQ0 运行中的进程在每次被抢占后，被降级到下一个低优先级队列中。类似于时间片

1的轮转法。

**缺点：**

若频繁有新进程进入系统，可能出现长进程饥饿。

**解决方法：**

1. 通过改变抢占次数解决，即越低优先级进程的允许执行之间越长。
2. 进程在队列中的等待时间超过一定量后，把它提升到一个优先级较高的队列。

**示例：**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 到达时间 | 服务时间 | 备注 |
| pro1 | 0 | 10 | 可能饥饿 |
| pro2 | 1 | 2 |  |
| pro3 | 2 | 2 |  |
| pro4 | 3 | 2 |  |
| pro5 | 4 | 2 |  |
| pro6 | 5 | 2 |  |

**pro1可能饥饿**

