

# 调度

## 背景介绍

调度器是操作系统的一部分，它决定计算机何时运行什么任务。通常，调度器能够暂停一个运行中的任务，将它放回到等待队列当中，并运行一个新任务，这一机制称为抢占（preemption）。抢占的实现往往需要通过硬件时钟（timer）定时发起中断（interrupt）信号，告知调度器一定时间周期已经过去，并由调度器决定下一个运行的任务。

调度器可能会针对不同的目标设计，例如：吞吐率最大化、响应时间最小化、延迟最小化或公平性最大化。在实践中，这些目标通常存在冲突；因此，调度器会实现一个权衡利弊的折中方案，根据用户的需求和目的，侧重以上一个或多个方面。

在实时（realtime）环境，例如工业上用于自动控制（如机器人）的嵌入式系统，调度器必须保证进程的调度不能超过最后期限——这是保持系统稳定运行的关键因素。

## 要求

本题要求设计一个面向单核的调度策略，并实现调度器对应接口。本题要求调度策略**尽可能达到实时系统的要求**（即所谓“准实时”调度），并根据接口实现的正确性与任务及时完成率进行评分。

## 任务假设

- 任务包含完成**截止时间**（deadline），调度器应尽可能在截止时间前完成任务。
- 任务分为高优先级与低优先级，调度器应倾向**优先完成高优先级任务**。
- 每个任务包括一段以上 CPU 计算与零或多段 IO 操作，分别使用计算机的 CPU 资源和 IO 资源。调度器可以执行任务  $T_{cpu}$  的 CPU 计算时，并行执行任务  $T_{io}$  的 IO 操作。
- 在一个任务使用 IO 资源时，不允许其同时使用 CPU 资源。
- 每个任务都以 CPU 计算开始与结束。

## 调度规则

**使用系统资源：**任何时刻，最多只有一个任务  $T_{cpu}$  使用系统 CPU 资源进行计算，最多只有一个任务  $T_{io}$  使用系统 IO 资源进行操作。系统 CPU 资源可以在任意时刻被调度器切换并执行新任务的 CPU 计算；系统 IO 资源必须完成当前 IO 操作后，才能执行新任务的 IO 操作。

**调度新任务：**调度器在新任务到达、任务结束、任务请求 IO 操作、任务结束 IO 操作、时钟中断到来时被唤醒并收到通知，并调用选手实现的策略接口决定接下来被调度的任务。策略将输出任务  $T'_{cpu}$  以抢占当前 CPU 资源。 $T'_{cpu}$  可以等于  $T_{cpu}$ ，即继续将 CPU 资源分配给旧任务； $T'_{cpu}$  可以为空，即将 CPU 资源空置。当不存在能够进行 CPU 计算的任务时，空置 CPU 资源是合理的。当 IO 资源空闲时，策略可输出任务  $T'_{io}$  以使 IO 资源服务新的任务。注意，因为 IO 资源无法实时切换，**当旧任务  $T_{io}$  未完成时，不能开始新的 IO 操作。**

## 子任务

本题共16个测试点：

1. 测试点1、2、3：任务特性较为相似，截止时间较为宽裕，优先级随机分布，任务随机出现；
2. 测试点4、5、6：在第1条基础上，任务特征差异较大；
3. 测试点7、8：在第2条基础上，任务特征分布随时间变化；

- 测试点9、10：在第2条基础上，截止时间较为紧张；
- 测试点11、12：在第2条基础上，具有相对紧张截止时间的任务倾向于拥有高优先级；
- 测试点13、14：在第2条基础上，任务在特定时刻会更加频繁出现；
- 测试点15、16：在第2条基础上，任务特征分布随时间变化，截止时间较为紧张，具有相对紧张截止时间的任务倾向于拥有高优先级，任务在特定时刻会更加频繁出现。

## 答题接口

本题要求选手实现调度策略 `policy` 接口，此函数将在上述描述的事件发生时被调用，此函数的输出将决定调度器接下来的操作。此处将描述该函数输入、输出参数语义，编程语言相关的细节将在后文具体描述。

`policy` 函数接收三个参数：

第一个参数为**事件列表**，包含此时刻同时发生的所有事件信息，绝大部分时候此列表长度为1。事件包括下列类型：

- 时钟中断到来 (`Timer`)；
- 新任务到达 (`TaskArrival`)，表示一个用户发起了新任务；
- 任务请求 IO 操作 (`IoRequest`)，表示一个任务需要进行 IO 操作；
- 任务结束 IO 操作 (`IoEnd`)，表示一个任务完成了一次 IO 操作，并需要使用 CPU 资源；
- 任务完成 (`TaskFinish`)，表示一个任务完成了它所有的 CPU 和 IO 操作。

除**事件类型**、**时间**外，与任务相关的事件信息还有相应的**任务信息**，包括**任务 ID**、**到达时间**、**截止时间**与**优先级**。

第二、三个参数为此时刻 CPU 与 IO 分别服务的任务的**任务 ID**。注意，当调度策略输出非法操作指令以及任务当前 CPU、IO 操作完成时，这两个参数可能会与上次 `policy` 指定的任务 ID 不同。

`policy` 函数输出调度策略的操作指令，即 CPU、IO 资源接下来分别服务任务的**任务 ID**。

根据调度规则，在 IO 资源未空闲时指定其他任务 ID 是非法的；在一个任务占用 IO 资源时，试图让其占用 CPU 资源也是非法的。进行任何非法操作会导致你在该测试点获得 0 分。

## 任务 ID 值的说明

所有任务的任务 ID 大于0。`policy` 函数输入参数中为0的任务 ID 代表对应资源空闲；输出操作指令中为0的 CPU 资源任务 ID 代表不使用 CPU 资源（即到下一个事件来临之前，CPU 将处于空闲状态），为0的 IO 资源任务 ID 代表不进行 IO 资源调度（即到下一个事件来临之前，IO 将保持当前状态）。

## C++接口

C++接口定义以下结构、函数。

```
1 struct Event {
2     enum class Type {
3         kTimer, kTaskArrival, kTaskFinish, kIoRequest, kIoEnd
4     };
5     struct Task {
6         enum class Priority { kHigh, kLow };
7
8         int arrivalTime;
9         int deadline;
10        Priority priority;
11        int taskId;
12    };
13 }
```

```
13
14     Type type;
15     int time;
16     Task task;
17 };
18
19 struct Action {
20     int cpuTask, ioTask;
21 };
22
23 Action policy(const std::vector<Event>& events, int currentCpuTask,
24               int currentIoTask);
```

请在相应文件中实现 `policy` 函数。其他辅助性结构、函数可定义、实现在同一文件中。本文件将使用 C++ 17 语言标准进行编译。

## 评分方式

### 正确性

考虑以下 FIFO 调度策略：维护任务队列  $Q$ ，当任务到达时将任务加入  $Q$ ，当且仅当上一任务完成时推出队首任务并开始执行。令 FIFO 调度下所有任务完成所需时间为  $t_{FIFO}$ ，当选手实现策略完成所有任务用时  $t$  小于等于  $t_{FIFO}$  时，认为选手策略实现正确。

### 性能指标：综合完成率

综合完成率  $r$  为高、低优先级任务在截止时间内及时完成率的加权平均。假设高优先级任务完成率为  $r_{hi}$ ，低优先级任务完成率为  $r_{lo}$ ，则综合完成率  $r$  满足：

$$r = 70\% \times r_{hi} + 30\% \times r_{lo}$$

### 总分

总分100分，其中 16 分为正确性得分：总共 16 输入文件，每个文件为 1 分。84 分为性能评比得分：每个测试点满分为 5.25 分，只有当选手得到正确性得分后才能得到性能分。设  $r_{\max}$  为所有选手本测试点的  $r$  的最大值，该测试点的性能得分为：

$$5.25 \times \frac{r}{r_{\max}}$$

## 提示

本题通过模拟方式评估选手调度策略性能，请选手不要以现实世界的时间单位来认知事件的时间（`time`）属性。

模拟过程中的时钟中断间隔长度固定，且大于 1。