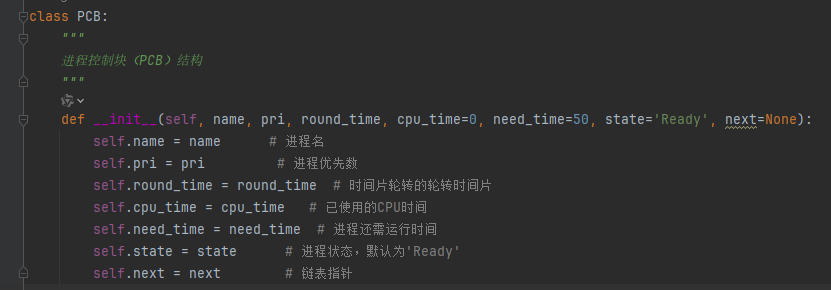
8.1代码

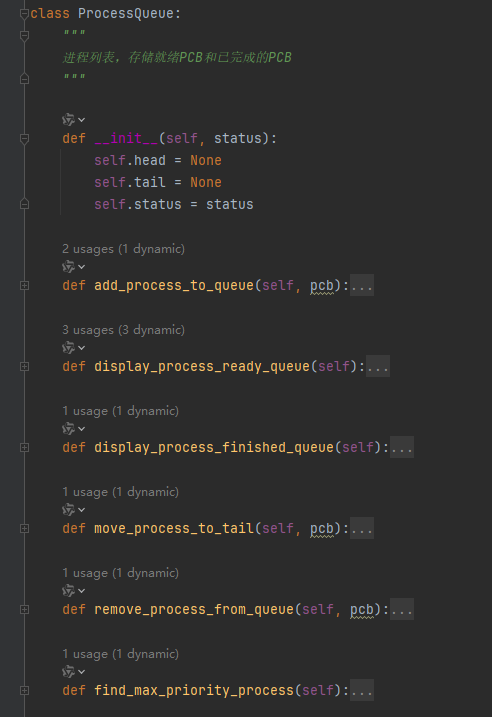
1. 进程控制块PCB类定义

。

这个PCB（Process Control Block）类定义了一个进程控制块的结构，它是操作系统中管理进程状态和资源分配的基本数据结构。每个PCB对象代表一个进程，包含了关于进程的各种信息。下面是各个属性的解释：

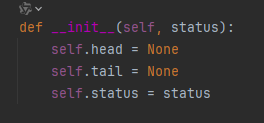
* 1. name: 进程的名称，通常是一个字符串，用于标识进程，这里一般是‘P1’...。
  2. pri: 进程的优先级，用于在动态优先调度算法决定哪个进程先执行，pri大的先执行。
  3. round\_time: 表示进程在一次调度周期内可以使用的CPU时间。
  4. cpu\_time: 这个进程已经使用过的CPU时间，用于跟踪进程的执行情况。
  5. need\_time: 进程还需要多少时间才能完成，如果未提供，则默认为50。
  6. state: 进程的状态，如'Running'、'Ready'、'Blocked'等，用于表示进程当前的执行状态。
  7. next: 链表指针，用于连接多个PCB对象，形成一个链表，这样可以方便地组织和遍历进程队列

1. 进程队列ProceseeQueue类定义

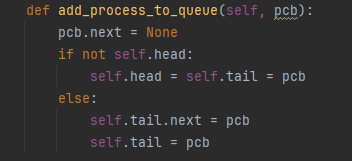


ProcessQueue 类是一个用于存储和管理进程控制块（PCB）的链表。根据 status 参数，一共会出现2个ProcessQueue对象，一个用于控制ReadyQueue、一个用于控制FinishQueue。以下是类中各个方法的简要介绍：

1. \_\_init\_\_(self, status)：构造函数，初始化链表的头节点 head 和尾节点 tail 为 None，并设置队列的状态 status。



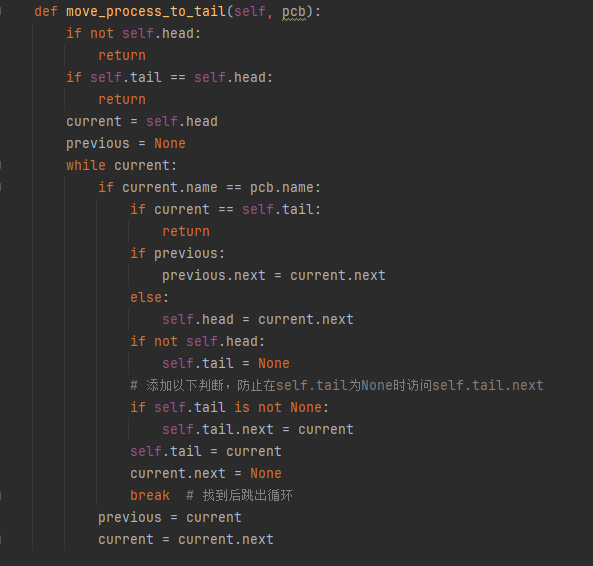
1. add\_process\_to\_queue(self, pcb)：向链表中添加一个新的 PCB。如果链表为空，head 和 tail 指向新的 PCB；否则，将 PCB 添加到链表的末尾。



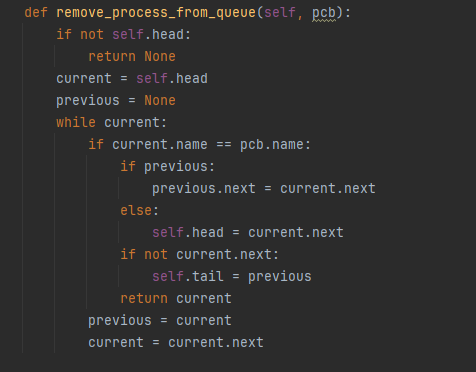
1. display\_process\_ready\_queue(self) 和 display\_process\_finished\_queue(self)：分别用于打印就绪队列和完成队列的 PCB 列表。前者会显示Need\_time和pri，后者会显示Cpu\_time和pri



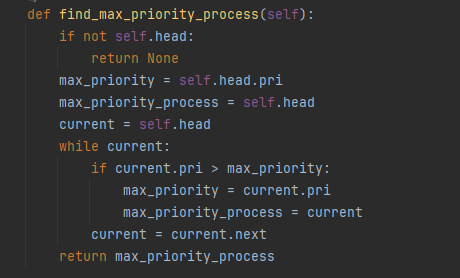
1. move\_process\_to\_tail(self, pcb)：将具有指定名称的 PCB 移动到链表的尾部。这个函数是在一个时间片结束后将此次执行的进程放到最后。防止其他进程饿死。



1. remove\_process\_from\_queue(self, pcb)：从链表中移除具有指定名称的 PCB。返回移除的 PCB。这个函数是在一个时间片结束后，这个进程finished，就需要把其从ReadyQueue转移到FinishedQueue。

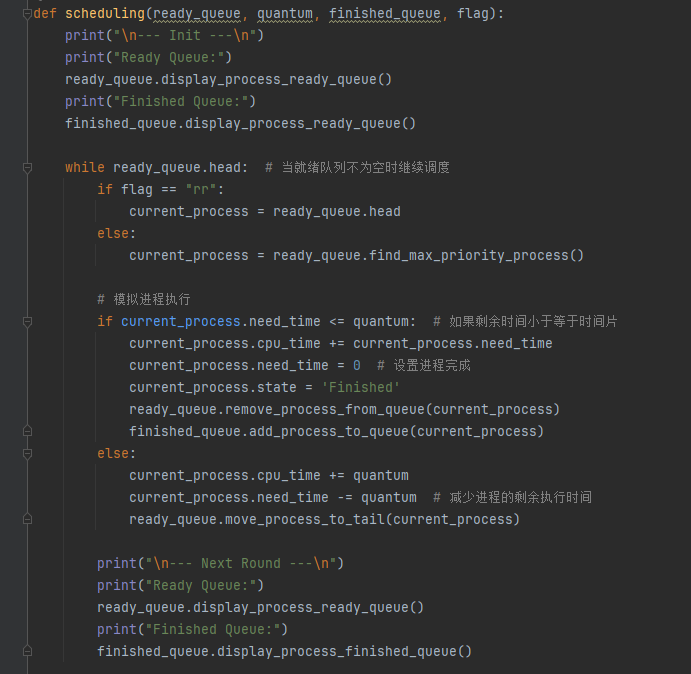


1. find\_max\_priority\_process(self)：查找链表中优先级最高的 PCB。如果链表为空，返回 None；否则，返回优先级最高（数值最大）的 PCB。这是特别针对于动态优先级队列。



这个类提供了对进程队列的基本操作，如添加、显示、移动和删除进程，以及查找优先级最高的进程。这些操作对于模拟操作系统中的调度算法非常有用。

1. 调度函数



scheduling 函数是一个模拟操作系统调度过程的函数，它接受四个参数：

1. ready\_queue：一个 ProcessQueue 对象，表示当前的就绪队列。
2. quantum：一个整数，表示时间片大小，用于时间片轮转调度（Round Robin, RR）。
3. finished\_queue：另一个 ProcessQueue 对象，用于存储已经完成的进程。
4. flag：一个字符串，指示调度策略，如果是 "rr"，则使用时间片轮转，否则使用优先级调度。

函数的主要步骤如下：

1. 初始化：在开始调度之前，打印当前的就绪队列和完成队列的内容。
2. 调度循环：当就绪队列非空时，持续进行调度。如果 flag 是 "rr"，选择就绪队列的第一个进程；否则，选择优先级最高的进程。轮转调度和动态优先级调度就是不同于这个函数。
3. 模拟进程执行：根据进程的剩余时间与时间片的关系进行处理：
4. 如果剩余时间小于等于时间片，进程完成，更新其 CPU 时间，设置为完成状态，从就绪队列中移除，并将其添加到完成队列。
5. 否则，进程执行了时间片的时间，更新其 CPU 时间和剩余时间，然后将其移动到就绪队列的末尾。
6. 输出状态：每次调度循环结束后，再次打印当前的就绪队列和完成队列内容，以便用户观察调度过程。

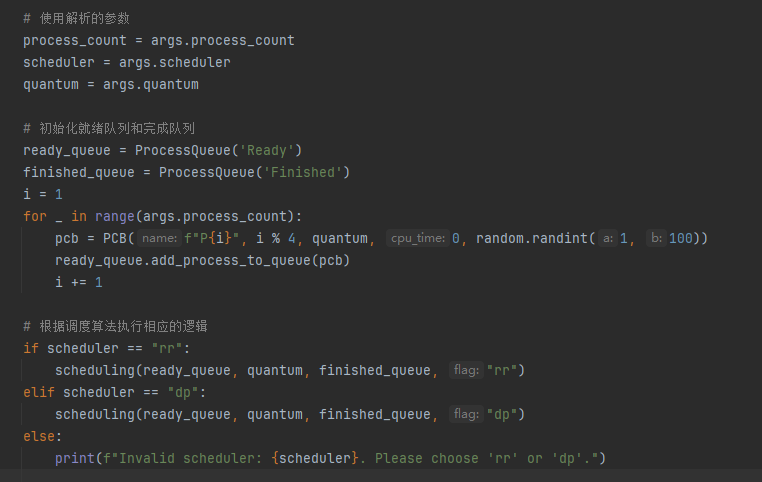
这个函数模拟了操作系统调度器的行为，可以根据不同的调度策略（时间片轮转或优先级调度）来处理进程，并展示调度过程中的状态变化。

1. 命令行参数输入



这段代码使用Python的argparse模块来处理命令行参数，目的是让用户在运行程序时能够自定义输入参数。具体来说，它定义了三个参数：

1. -c 或 --process\_count：这是可选参数，用于指定进程的数量。用户可以通过在命令行中添加-c后面跟一个整数来设置进程数，例如-c 5表示有5个进程。
2. -s 或 --scheduler：也是一个可选参数，用于选择调度算法。用户可以选择rr（轮转调度）或dp（动态优先级调度）。例如，-s rr会选择轮转调度。
3. -q 或 --quantum：当选择轮转调度时，这个参数是必需的，用于设置时间片的长度。用户需要提供一个整数，例如-q 10表示每个进程在被抢占前可以执行10个时间单位。
4. main



这段代码主要完成了以下任务：

1. 使用解析的参数：

* process\_count = args.process\_count: 获取命令行参数中指定的进程数量。
* scheduler = args.scheduler: 获取用户选择的调度算法（轮转或动态优先级）。
* quantum = args.quantum: 获取时间片长度，仅在轮转调度时使用。

1. 初始化队列：

* 创建一个名为ready\_queue的ProcessQueue实例，表示就绪队列，状态设为'Ready'。
* 创建一个名为finished\_queue的ProcessQueue实例，表示完成队列，状态设为'Finished'。
* 使用一个循环，根据process\_count创建相应数量的PCB对象。i % 4用于为进程分配0-3的优先级，random.randint(1, 100)生成一个1到100之间的随机数作为每个进程的需时间，然后将这些PCB对象添加到就绪队列中。

1. 调度执行：

* 根据用户选择的调度算法执行相应的调度函数scheduling。如果选择了rr，则传递"rr"作为标志执行时间片轮转调度；如果选择了dp，则传递"dp"作为标志执行动态优先级调度。如果用户提供的调度算法无效，程序会输出错误消息。