Operating System CH5

Susie Glitter

2025 年 7 月 7 日

注：本次实验使用了 VMware 中的 ubuntu-16.04.6-desktop

1 任务一：实现五种调度算法

1.1 FCFS 算法的实现

FCFS 算法根据到达时间顺序进行执行，只需要每次将队首的任务执行并且出队即可，执行时间直接设置为任务的总时长。

1.2 SJF 算法的实现

SJF 算法对于同时到达的任务，优先执行最短的任务，我们由队尾向队首进行遍历，获得到其中最早到达且时间最短的进行执行并且出队，执行时间同样为该任务的总时长。

1.3 RR 算法的实现

RR 算法每次执行时间最多为 QUANTUM 长度，在这里为 10，我们每次执行队首的任务并出队，执行时间为 QUANTUM 与任务剩余时间之间的较小值，若任务没有执行完，修改剩余时间后再将其追加到队尾，等待下一次轮询执行剩余部分。

1.4 Priority 算法的实现

Priority 算法对于同时到达的任务，优先执行优先级最高的一个进行执行，每次由队尾遍历至队首，找出优先级最高的任务进行执行并且出队，执行时间为该任务的总时长。

1.5 Priority_RR 算法的实现

Priority_RR 算法结合前两个算法，每次由队尾向队首遍历，获得优先级最高且靠近队首的一个任务并出队，执行时间为 QUANTUM 与该任务剩余时间的较小值，若任务没有执行完，修改剩余时间后再将其追加到队尾，等待下一次轮询执行剩余部分。

2 任务二：使用原子操作

在新建线程时需要申请对应的 tid，使用原子操作对某个值进行递增，直到来到可用的 tid，可以避免多个核获取到同一个 tid 造成冲突。可以使用`\_\_sync_fetch_and_add()`函数进行其中的原子变量的自增操作。

`AtomicInteger`类继承自`Number`类，且实现了一些原子操作。例如原子变量的自增、自减、更新等原子操作。举个例子，为了实现上面提到的线程分配，我们可以使用`int incrementAndGet()`自增一个原子变量并且返回值，作为新获得的 tid 的值，从而避免发生 race condition

3 任务三：计算五种调度算法的相关数据

为方便进行对比，以下数据均基于`schedule.txt`进行计算，具体数据见文件。

	平均周转时间	平均响应时间	平均等待时间
FCFS	73.125	94.375	73.125
SJF	61.25	82.5	61.25
RR	35.0	128.75	107.5
Priority	75.0	96.25	75.0
Priorities_RR	71.875	105.0	83.75