操作系统作业 4

1. Consider the deadlock situation that can occur in the dining-philosophers problem when the philosophers obtain the chopsticks one at a time. Discuss how the four necessary conditions for deadlock hold in this setting. Describe a deadlock-free solution, and discuss which necessary conditions are eliminated in your solution.

解：　当哲学家一次只拿一根筷子时，有可能出现死锁状况。例如有n个哲学家，某一时刻他们同时感到饥饿并停止思考，按照进食步骤，他们试图拿起两边的筷子。而一次只允许拿一根筷子，此时所有的筷子都是空闲的，没有什么能阻止每个哲学家拿起自己左边的一根筷子。恰好此时每个哲学家都拿到了一根筷子并开始等待右边筷子，这个等待只能无限的推迟。产生死锁。

死锁的四个必要条件分别为：互斥，占有并等待，非抢占，循环等待。在上述处理过程中，这四个条件均被满足：

互斥：筷子是非共享资源，一次只有一个哲学家（进程）使用。如果另一个哲学家（进程）申请该资源，那么申请进程必须等待到资源释放。

占有并等待：每个哲学家（进程）都占有一个资源（左边筷子），并同时都在等待另一个资源（右边筷子），而右边筷子被其他哲学家所占有。

非抢占：资源不能被抢占，只能在进程完成任务后自愿释放。哲学家不可以从其他的哲学家手中抢走筷子，只能等其他哲学家吃完放下筷子。

循环等待：有一组哲学家$\{T_1, T_2, \cdots\cdots T_n\}$，每一个哲学家等待的资源都被下一个哲学家所占有。

这四个条件同时成立，出现了死锁状况。

利用以下方法可以解决哲学家就餐问题的死锁情况，方案如下：

先加入以下限制：只有当一个哲学家左右两根筷子都可用且该哲学家处于临界区内时，他才能拿起筷子。因此需要区分哲学家所处的三个状态，引入如下数据结构：

$$\text{enum }\{\text{THINKING, HUNGRY , EATING}\}\text{state[n];}$$

哲学家i只有在两个邻居都不在就餐时才能设置$\text{state[i]} = \text{EATING}$. 吃完后他将设置$\text{state[i]} = \text{THINKING}$.

为此，还需设置 $\text{condition self[n];}$ 这让哲学家在感到饥饿又拿不到筷子时可以延迟自己。同时，哲学家拿起筷子的行为必须在临界区内进行，当有任意一个哲学家在改变筷子的使用状态时，其他哲学家都无法改变筷子状态，必须等在临界区内的哲学家拿起左右两只筷子或放下左右两只筷子。

该方案破坏了死锁条件中的占有并等待条件，哲学家不会出现只拿到一根筷子的情况。这种方法可以避免死锁，但是哲学家可能会一直饥饿。

2. Consider the exponential average formula used to predict the length of the next CPU burst. What are the implications of assigning the following values to the parameters used by the algorithm?
a. α = 0 and τ_0 = 100 milliseconds
b. α = 0.99 and τ_0 = 10 milliseconds

解：下次的 CPU 执行通常预测为以前 CPU 执行的测量长度的指数平均。

设t_n为实际上第n个 CPU 执行长度，τ_{n+1}为下次 CPU 执行的预测值。

对于$0 \le \alpha \le 1$，定义：

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$$

参数α控制了最近和过去历史在预测中所占的权重。

情况a：此时$\alpha = 0$，最近对预测结果没有影响，预测完全由过去历史所决定。

由定义式可知：$\tau_{n+1} = \tau_n = \cdots\cdots = \tau_0 = 100\ ms$

情况b：此时$\alpha = 0.99$，$\tau_{n+1} = 0.99t_n + 0.01\tau_n$，因此过去历史在预测中所占比例极低，几乎没有影响，而最近的 CPU 执行才比较重要。预测初始值从$\tau_0 = 10\ ms$开始，逐步带入定义式即可求出下次预测的值。

3. Consider the following set of processes, with the length of the CPU burst time given in milliseconds:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF (non-preemptive), non-preemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1).

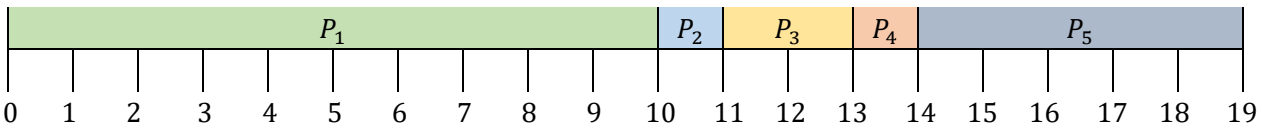
b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

c. What is the waiting time of each process for each of these scheduling algorithms?

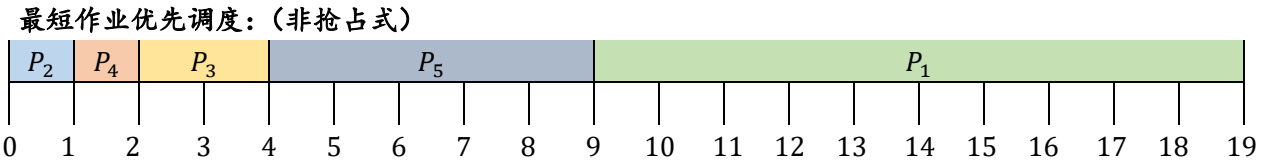
d. Which of the algorithms results in the minimum average waiting time (over all processes)?

解：画出这些调度算法的甘特图：（单位：ms）

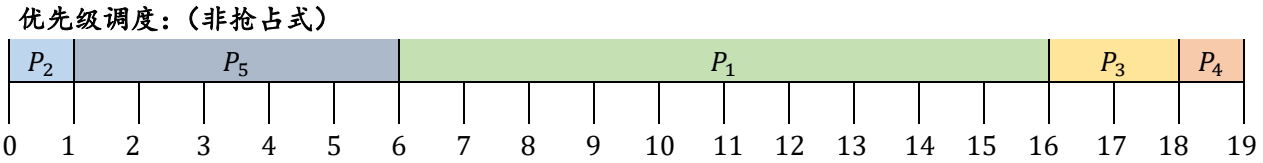
先到先服务调度：



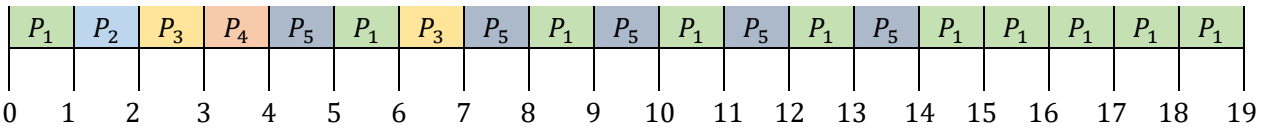
最短作业优先调度：（非抢占式）



优先级调度：（非抢占式）



轮转调度：（每次完成 1 个时钟）



再求每个进程在不同调度方案下的周转时间：（从提交到完成的时间）

周转时间	P_1	P_2	P_3	P_4	P_5
先到先服务调度	10 ms	11 ms	13 ms	14 ms	19 ms
最短作业优先调度	19 ms	1 ms	4 ms	2 ms	9 ms
优先级调度	16 ms	1 ms	18 ms	19 ms	6 ms
轮转调度	19 ms	2 ms	7 ms	4 ms	14 ms

求每个进程在不同调度方案下的等待时间：（在就绪队列中等待所花的**时间之和**）

等待时间	P_1	P_2	P_3	P_4	P_5
先到先服务调度	0 ms	10 ms	11 ms	13 ms	14 ms
最短作业优先调度	9 ms	0 ms	2 ms	1 ms	4 ms
优先级调度	6 ms	0 ms	16 ms	18 ms	1 ms
轮转调度	9 ms	1 ms	5 ms	3 ms	9 ms

再计算每种调度的平均等待时间：

	平均等待时间
先到先服务调度	9.6 ms
最短作业优先调度	3.2 ms
优先级调度	8.2 ms
轮转调度	5.4 ms

因此在这种情况下最短作业优先调度算法所消耗的平均等待时间最短。

4. Which of the following scheduling algorithms could result in starvation?

e. First-come, first-served

f. Shortest job first

g. Round robin

h. Priority

解：最短作业优先调度算法和优先级调度算法有可能会导致进程饥饿

- **最短作业优先调度算法：**一个 CPU 密集型进程可能长时间得不到调度，会导致饥饿状态发生。尤其是在抢占型最短作业优先调度算法中，一个很长的 CPU 作业可能被数次抢占，也可能一直得不到进行。
- **优先级调度：**就绪但是在等待 CPU 的进程认为是阻塞的，优先级调度算法可能让某个低优先级进程无穷等待 CPU. 为防止这种情况的发生，可以使用老化方案。老化逐渐增加在就绪队列中等待很长时间的低优先级进程的优先级，确保在一定时间内最低优先级的进程也能得到执行。

5. Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context-switching overhead is 0.1millisecond and that all processes are long-running tasks. Describe is the CPU utilization for a round-robin scheduler when:
   i. The time quantum is 1 millisecond
   j. The time quantum is 10 milliseconds

解：在十个I/O密集型任务中，由题意，每个任务在每个轮转周期需要 1 ms 的 CPU 时间和 10 ms 的外设读写时间。而只有这些任务需要用到 CPU 且轮转到该任务执行时，CPU 才由这个任务执行使用。

• **当时间片＝ 1 ms时：** 恰好一次轮转周期中每个I/O密集型任务都恰好使用 CPU 完成了它的 CPU 执行部分，而总共十一个任务，保证了下一次轮转周期来临之前该任务已经读写I/O设备完毕，进入了等待 CPU 的就绪队列中。因此，对于 CPU 来说。无论是I/O密集型任务还是 CPU 密集型任务，在每个轮转周期内都执行1 ms，每次切换任务时都需要一次上下文切换开销。在一个轮转周期内计算：

$$\text{CPU 利用率} = \frac{1 * 10 + 10}{1 * 10 + 10 + 0.1 * 20} = 90.91\%$$

• **当时间片＝ 10 ms时：** 每一个I/O密集型任务只需要 1 ms 的 CPU 使用时间。时间片为 10 ms 时，CPU 密集型任务恰好在一个轮转周期内完成它的 CPU 执行部分；而十个I/O密集型任务在一个时间片内恰好执行完毕（每1ms切换一个，由于使用完成，直接切换）。因此，在一个轮转周期内计算：

$$\text{CPU 利用率} = \frac{1 * 10 + 10}{1 * 10 + 10 + 0.1 * 11} = 94.79\%$$

6. Give an example to illustrate under what circumstances rate-monotonic scheduling is inferior to earliest-deadline-first scheduling in meeting the deadlines associated with processes?

解：在满足不超过任何一轮的最后期限的情况下，如果不考虑切换开销，速率单调调度与最早截止期限优先调度的 CPU 使用率是完全一样的，因为他们都是最优调度方案。

简单证明如下：

没有任何一轮有进程超过最后期限，假设有i个进程，每个进程的周期为$p_1, p_2 \cdots\cdots p_i$，每个进程需要的 CPU 执行时间为$t_1, t_2, \cdots\cdots t_i$.这些时间必然在时间轴上$p = p_1 * p_2 * \cdots\cdots = \prod p_i$处交汇。所以这个长度可以看作是一个大周期，可以在这一段中计算 CPU 的使用率：

假设进程P_j运行n_j轮，总的运行时间$= \sum n_j * t_j$　而进程间 CPU 使用时间不能重叠

因此两种调度 CPU 使用率均为：$\frac{\sum n_j * t_j}{\prod p_i}$

如果考虑切换开销，很多时候最早截止期限优先调度好于速率单调调度。例如：

有两个进程，$p_1 = 50$ ms，$p_2 = 100$ ms，$t_1 = 20$ ms，$t_2 = 35$ ms

运用速率单调调度的甘特图如下所示：在100 ms 内，速率单调调度比最早截止期限优先调度要多了两次进程间切换。