OS hw3

1. T1

> What is race condition and how to address it?

- Race Condition：竞态条件，是指多个进程或线程在并发执行时，由于对共享资源的访问顺序不确定或不正确，导致最终结果依赖于执行顺序的一种情况。具体来说，当多个进程或线程尝试同时访问和修改共享数据时，会发生竞态条件
- 解决方法：
    1. 互斥锁：当一个进程或线程获得了互斥锁后，其他进程或线程必须等待，直到该进程或线程释放锁
    2. 信号量：信号量可以表示可以同时访问共享资源的进程或线程数量，并根据需要进行增加或减少
    3. 条件变量：条件变量允许线程等待某个条件满足后再继续执行
    4. 原子操作：原子操作是不可中断的操作，要么完全执行，要么不执行
    5. 并发编程模型：使用适当的并发编程模型，如消息传递、事件驱动等，来减少竞态条件的发生

2. T2

> List all the three requirements of the entry and exit implementation when solving the critical-section problem.

- 互斥性：在任何时刻，只允许一个进程或线程进入临界区
- 空闲让进：如果没有进程在临界区执行，并且有进程请求进入临界区，那么只有那些没有违反第一个要求的进程可以选择进入临界区
- 有界等待：对于进程请求进入临界区的时间，应该存在一个上限。这意味着，如果一个进程想要进入临界区，但其他进程已经进入了临界区，那么该进程必须在有限的时间内能够进入临界区，而不是无限期地等待

3. T3

> For Peterson's solution, prove that it satisfies the three requirements of a solution for the critical section problem. Analyze whether strict alternation satisfies all the requirements

- 对于Peterson算法，可以证明它满足临界区问题的三个要求：

1. 互斥性：Peterson算法通过使用两个标志位来实现互斥性。每个进程在进入临界区之前，都会设置自己的标志位，表明它希望进入临界区。如果另一个进程的标志位已经被设置，即另一个进程正在临界区执行，那么当前进程会等待，直到另一个进程退出临界区。这样确保了在同一时间只有一个进程可以进入临界区，满足互斥性。
2. 空闲让进：Peterson算法通过轮流设置两个进程的标志位来实现进程进入。如果两个进程都希望进入临界区，但其中一个进程的标志位已经被设置，那么另一个进程会等待。当且仅当另一个进程退出临界区后，才会设置自己的标志位并进入临界区。这确保了只有那些没有违反互斥性的进程可以进入临界区，满足进程进入要求。
3. 有界等待：Peterson算法通过使用"turn"变量来实现有界等待。每个进程在进入临界区之前，都会将"turn"设置为另一个进程的标识。这意味着进程会等待另一个进程进入临界区并完成，然后才会自己进入。这确保了等待进入临界区的进程不会无限期地被阻塞，满足有界等待要求。

- 对于严格交替（strict alternation）方法，两个进程按照固定的顺序轮流进入临界区。只有当另一个进程退出临界区后，另一个进程才能进入。这样确保了只有一个进程在临界区执行，满足互斥性。进程进入要求通过固定的轮流顺序来实现，确保每个进程都有机会进入临界区。有界等待要求是通过进程的轮流顺序和进入退出的约定来实现的，确保等待进入临界区的进程不会无限期地被阻塞，因此，也满足临界区问题的三个要求

4. T4

> What is deadlock? List the four requirements of deadlock

- 死锁是指在多个进程或线程中，每个进程都在等待其他进程持有的资源，导致所有进程都无法继续执行的一种情况。换句话说，当一组进程中的每个进程都无法获得所需的资源，并且又不释放自己已经占有的资源时，就会发生死锁。
- 四个条件：

1. 互斥条件：每个资源一次只能被一个进程持有或访问。
2. 请求与保持条件：进程在持有至少一个资源的同时，又请求额外的资源，而这些额外的资源被其他进程占有。即进程在请求资源时不会释放已经持有的资源。
3. 不可剥夺条件：已经分配给进程的资源不能被其他进程强制性地剥夺，只能由占有资源的进程显式地释放。
4. 循环等待条件：一组进程形成一个循环等待的关系，每个进程都在等待下一个进程所持有的资源。

5. T5

Consider the following snapshot of a system：

	Allocation	*Max*
	A B C D	*A B C D*
T_0	1 2 0 2	4 3 1 6
T_1	0 1 1 2	2 4 2 4
T_2	1 2 4 0	3 6 5 1
T_3	1 2 0 1	2 6 2 3
T_4	1 0 0 1	3 1 1 2

Using the banker's algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe

a. Available = (2, 2, 2, 3)

b. Available = (4, 4, 1, 1)

c. Available = (3, 0, 1, 4)

d. Available = (1, 5, 2, 2)

a

	work	allo	need	w+a	T/F
T4	2223	1001	2111	3224	T
T0	3224	1202	3114	4426	T
T1	4426	0112	2312	4538	T
T2	4538	1240	2411	5778	T
T3	5778	1201	1422	6979	T

b

	work	allo	need	w+a	T/F
T4	4411	1001	2111	5412	T
T1	5412	0112	2312	5524	T
T0	5524	1202	3114	6726	T
T2	6726	1240	2411	7966	T
T3	7966	1201	1422	8(11)67	T

c

不安全，因为B的available是0，而所有max-allocation都大于0，所以第一步找不到能运行的线程

d

	work	allo	need	w+a	T/F
T3	1522	1201	1422	2723	T
T1	2723	0112	2312	2835	T
T2	2835	1240	2411	3(10)75	T
T0	3(10)75	1202	3114	4(12)77	T
T4	4(12)77	1001	2111	5(12)78	T

6. T6

> What is semaphore? Explain the functionalities of semaphores.

- 信号量是一种同步机制，用于控制多个线程或进程之间的访问共享资源。
- func：
  1. 除了初始化之外，只能通过两个标准原子操作访问，P和V
     - P：当线程或进程需要访问共享资源时，它会尝试获取信号量。如果信号量的值大于0，表示资源可用，线程或进程可以继续执行。如果信号量的值为0，表示资源不可用，线程或进程将被阻塞，直到信号量的值大于0
     - V：当线程或进程释放共享资源时，它会调用V操作来增加信号量的值。这会唤醒等待该资源的其他线程或进程，使它们可以继续执行
  2. 可以用来解决并发环境下的临界区问题，确保多个线程或进程之间的顺序执行和互斥访问
  3. 通过使用信号量，可以实现线程或进程之间的同步和互斥，避免竞态条件和资源冲突问题

7. T7

> Please use semaphore to provide a deadlock-free solution to address the dining philosopher problem

餐厅哲学家问题：五位哲学家共享圆桌上的五个餐叉。每位哲学家需要交替地进行思考和进餐，但他们只能使用自己左右两侧的餐叉。避免死锁

使用信号量来解决餐厅哲学家问题的一种方法：

1. 创建一个与哲学家数量相等的信号量数组，每个元素代表一个餐叉。初始时，所有餐叉的信号量值为1，表示可用。
2. 每个哲学家线程的执行过程如下：
    - 思考阶段：哲学家线程进行思考，不需要获取餐叉。
    - 进餐阶段：哲学家线程尝试获取左手和右手的餐叉，通过P操作来获取餐叉。
    - 如果左右两侧的餐叉都可用，哲学家线程将两个餐叉的信号量值都减1，表示占用餐叉。
    - 如果某个餐叉不可用，哲学家线程将被阻塞，直到餐叉可用。
    - 进餐完成后，哲学家线程释放左手和右手的餐叉，通过V操作来增加餐叉的信号量值。

通过这种方式，可以避免死锁的发生，因为哲学家线程只会获取两个餐叉中的一个，并且总会有一个餐叉是可用的。

8. T8

> Consider the following set of processes, with the length of the CPU burst time given in milliseconds:
>
> | Process | Burst Time | Priority |
> | --- | --- | --- |
> | P1 | 10 | 3 |
> | P2 | 1 | 1 |
> | P3 | 2 | 3 |
> | P4 | 1 | 4 |
> | P5 | 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 (nonpreemptive), nonpreemptive 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)?
>
> e) Illustrate the pros and cons of the algorithms: FCFS, SJF, priority scheduling and RR

- a) 使用不同调度算法的四个甘特图

FCFS调度算法的甘特图：

P1	**P2**	**P3**	**P4**	**P5**
10	11	13	14	19

SJF调度算法的甘特图：

P2	**P4**	**P3**	**P5**	**P1**
1	2	4	9	19

非抢占式优先级调度的甘特图：

P2	**P5**	**P1**	**P3**	**P4**
1	6	16	18	19

时间片轮转调度的甘特图：

P1	P2	P3	P4	P5
1	2	3	4	5
6		7		8
9				10
11				12
13				14
15				
16				
17				
18				
19				

- b) 每个调度算法下每个进程的周转时间：

FCFS调度算法的周转时间：
P1: 10 - 0 = 10
P2: 11 - 0 = 11
P3: 13 - 0 = 13
P4: 14 - 0 = 14
P5: 19 - 0 = 19

SJF调度算法的周转时间：
P1: 19 - 0 = 19
P2: 1 - 0 = 1
P3: 4 - 0 = 4
P4: 2 - 0 = 2
P5: 9 - 0 = 9

非抢占式优先级调度的周转时间：
P1: 16 - 0 = 16
P2: 1 - 0 = 1
P3: 18 - 0 = 18
P4: 19 - 0 = 19
P5: 6 - 0 = 6

时间片轮转调度的周转时间：
P1: 19 - 0 = 19
P2: 2 - 0 = 2
P3: 7 - 0 = 7
P4: 4 - 0 = 4
P5: 14 - 0 = 14

- c) 每个调度算法下每个进程的等待时间：

FCFS调度算法：

	P1	**P2**	**P3**	**P4**	**P5**
等待时间	0	10	11	13	14

SJF调度算法：

	P1	**P2**	**P3**	**P4**	**P5**
等待时间	9	0	2	1	4

非抢占式优先级调度算法：

	P1	**P2**	**P3**	**P4**	**P5**
等待时间	6	0	16	18	1

时间片轮转（时间片=1）调度算法：

	P1	**P2**	**P3**	**P4**	**P5**
等待时间	9	1	5	3	9

- d)每个算法的平均等待时间：

  FCFS：（0+10+11+13+14）/5=9.6

SJF：（9+0+2+1+4）/5=3.2

非抢占式优先级调度：（6+0+16+18+1）/5=8.2

RR: (9+1+5+3+9)/5=5.4

因此SJF调度算法有最小的平均等待时间

- e）

各种算法优缺点：

FCFS：

优点：

- 简单易实现
- 公平性较高，按照进程到达的顺序进行调度

缺点：

- 平均等待时间较长，不考虑作业的执行时间
- 无法适应长作业与短作业的差异，可能导致短作业长时间等待

SJF：

优点：

- 平均等待时间较短，优先执行短作业
- 提高了系统的吞吐量

缺点：

- 难以准确预测作业的执行时间，可能导致长作业长时间等待
- 可能会发生饥饿现象，即某些长作业长时间得不到执行

非抢占式优先级调度：

优点：

- 可以根据作业的优先级进行调度，满足不同作业的优先级需求
- 可以提高紧急任务的响应速度

缺点：

- 可能导致低优先级作业长时间等待，出现饥饿现象
- 静态优先级调度可能不适应动态变化的系统需求

时间片轮转：

优点：

- 公平性较高，每个作业都有机会被执行
- 可以及时响应交互式任务，避免长时间等待

缺点：

- 时间片过长可能导致长作业的响应时间较长
- 时间片过短可能会引入较大的上下文切换开销

9. T9

> Illustarte the key ideas of rate-monotonic scheduling and earliestdeadline-first scheduling. 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?

- 速率单调调度的核心思想：任务的周期性执行频率与其优先级成反比。具有更短周期的任务具有更高的优先级，因此在调度时优先考虑执行周期更短的任务
- 最早截止时间优先调度的核心思想：任务的截止时间越早，优先级越高。在调度时，优先考虑具有最早截止时间的任务。
- 例子（速率单调调度何时不如最早截止时间优先调度来满足任务的截止时间）：

1. 两个任务：A和B
2. A的周期为10s,执行时间为6s；B的周期为20s,执行时间为10s
3. 根据速率单调调度算法，A的优先级高于B。因此，在调度时，A将先执行，6s后，执行B4s，A任务带来执行A6s，再执行B4s，此时已经到B任务deadline20s，而B任务只执行了8s，没有执行完成

4. 根据最早截止时间优先调度算法，在0时刻，A的deadline更早，先执行A6s，，然后B4s，在10时刻，此时A和Bdeadline均为20，所以继续执行B6s，完成B，此时A和B都在deadline前完成