操作系统 作业四

PB18071496 李昱祁

一． 答：

本例中的死锁条件：

a. 互斥：哲学家不能从其他人手中拿到筷子。即筷子某一时刻只能在一个人手中被使用
b. 占有和等待：当一个哲学家成功拿起了第一根筷子,却请求拿起第二根筷子失败时，他(相当于一个进程)便符合占有且等待的条件
c. 无优先权：筷子不能被抢占。任一位哲学家都不能从其他哲学家手中获得一根正在被使用的筷子。
d. 循环等待：若每个哲学家都拿到了他右边（或左边）的筷子，同时等待他左边（或右边）的哲学家释放筷子，则这些哲学家之间组成了一个循环等待的关系

一个 deadlock-free 解决方案：

```
/* Shared object */
#define N 5
#define LEFT ((i+N-1) % N)
#define RIGHT ((i+1) % N)
int state[N];
semaphore mutex = 1;
semaphore s[N];
void test(int i);
enum state{HUNGRY,THINKING,EATING};
/* Section entry */
void take(int i)
{
    down(&mutex);
    state[i] = HUNGRY;
    test(i);
```

```
    up(&mutex);
    down(&s[i]);
}
/* Section enxit */
void put(int i) {
    down(&mutex);
    state[i] = THINKING;
    test(LEFT);
    test(RIGHT);
    up(&mutex);
}
/* helper function */
void test(int i)
{
    if(state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
    {
        state[i] = EATING;
        up(&s[i]);
    }
}
/* Main Function */
void philosopher(int i)
{
    think();
    take(i);
    eat();
    put(i);
}
```

（其中 up、down 为对信号量进行同步增减的函数；think 为定义的实现“哲学家思考”的函数）

此种解决方案解决了四个条件中的 2 个：

a. 占有和等待：由 test 函数的实现可知，只有当某一哲学家左右两边的哲学家都不处于 EARING 状态时，该哲学家才

能进入 EATING 状态，进而开始被分配筷子。即不会出现某一哲学家占有了一只筷子后开始进行等待的情况

b. 循环等待：由上分析知哲学家在等待其他哲学家时，自己不会作为等待的对象（因为他此时不会占有筷子），进而不会出现循环等待的情况

二． 答：

a. $\alpha = 0$ ， $\tau(0) = 100\,ms$

则 $\tau(n+1) = \tau(n)$，即下一个预测只与上一个预测有关

而 $\tau(0) = 100$ ms，所以每次预测的值都会是 100 ms

b. $\alpha = 0.99$ ， $\tau = 10$ ms

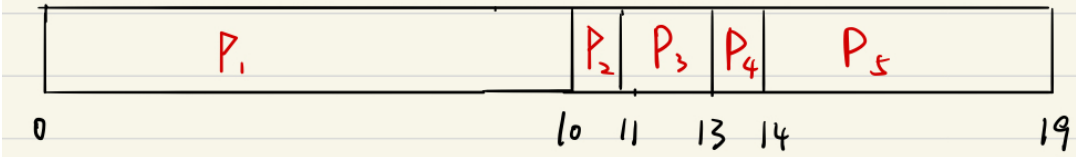
$\tau(n+1) = 0.99 * t(n) + 0.01\tau(n)$

即每一次预测 99%的权重都取决于上一次 cpu 实际执行的时间，而之前的预测几乎对下一次预测没有影响。之前的预测信息被认为是过时的、陈旧无关的。

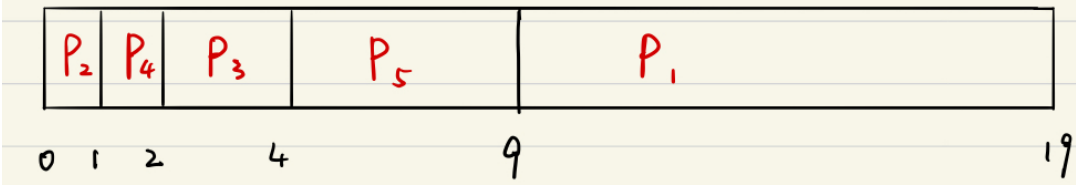
三． 答：

a.

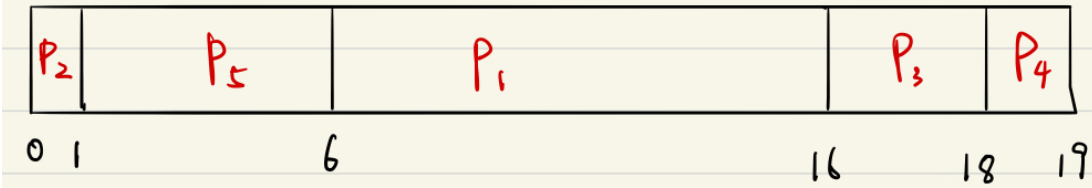
先到先服务（FCFS）：



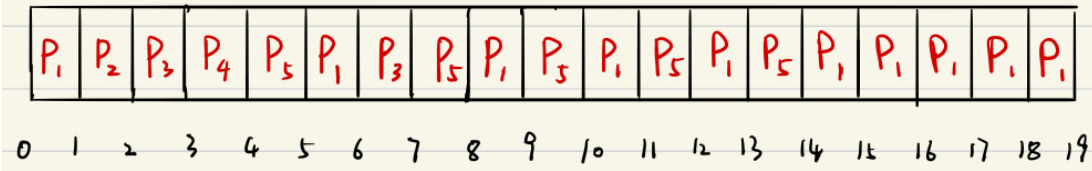
短作业优先（SJF）



优先级调度：



轮转调度：



b. 单位：ms

FCFS: P1 = 10, P2 = 11, P3 = 13, P4 = 14, P5 = 19

SJF: P1 = 19, P2 = 1, P3 = 4, P4 = 2, P5 = 9

Priority: P1 = 16, P2 = 1, P3 = 18, P4 = 19, P5 = 6

RR: P1 = 19, P2 = 2, P3 = 7, P4 = 4, P5 = 14

c. 单位：ms

FCFS: P1 = 0, P2 = 10, P3 = 11, P4 = 13, P5 = 14

SJF: P1 = 10, P2 = 0, P3 = 2, P4 = 1, P5 = 4

Priority: P1 = 6, P2 = 0, P3 = 16, P4 = 18, P5 = 1

RR: P1 = 9, P2 = 1, P3 = 5, P4 = 3, P5 = 9

d.

FCFS:	(0+10+11+13+14)/5	=	9.6	(ms)
SJF:	(10+0+2+1+4)/5	=	3.4	(ms)
Priority:	(6+0+16+18+1)/5	=	8.2	(ms)
RR:	(9+1+5+3+9)/5	=	5.4	(ms)

可见，上例中最短作业优先调度算法（SJF）可使平均等待时间最短

四． 答：

c. Shortest job first 和 d. Priority 两种调度算法可能导致饥饿

简要说明：

（1） SJF: 若不断有短作业进程到达，则某个需要长时间执行的进程会一直等待 cpu

（2） Priority: 若不断有高优先级进程到达，则低优先级的进程会无限等待 cpu

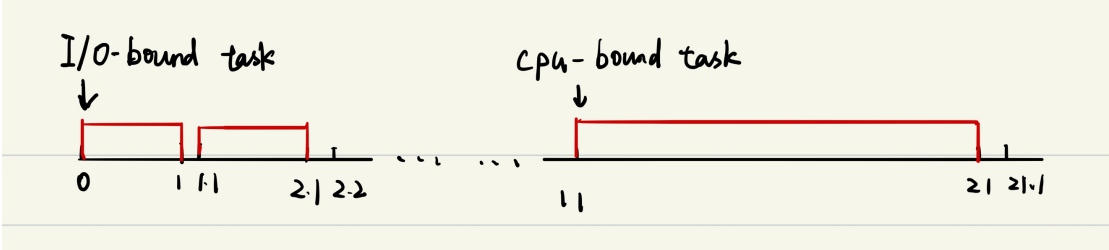
五． 答：

a. 时间片为 1ms，则每毫秒都要进行上下文切换

Cpu 利用率 = (1/(1+0.1)) * 100% = 90.9%

b. 每个 I/O 密集型任务执行 1ms 即切换，等待 I/O operation 的完成。现分析一个周期内的情况：

假设 10 个 I/O 密集型任务先执行，每个任务执行 1ms 即切换，考虑到切换用时 0.1ms，则从第 1 个任务开始执行算起，要用 10*(1+0.1) = 11(ms)，之后 cpu 密集型任务开始执行，用时 10ms，再进行一次上下文切换，第一个 I/O 密集型任务准备开始执行，如下图：



则 cpu 利用率 = （10*1 + 10）/ 21.1 = 94.79%

六． 答：

举例如下：

进程 P1 有周期 p1 = 50 和 t1 = 30 ；

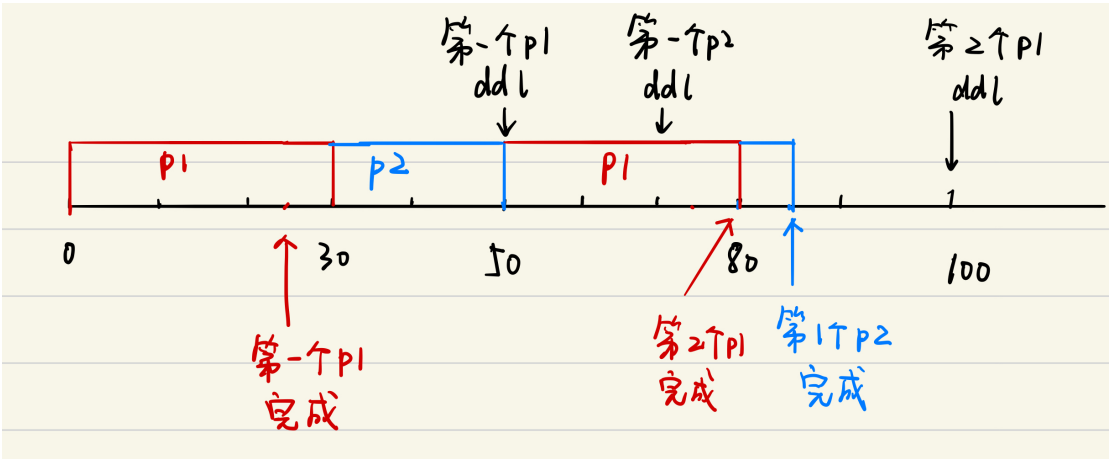
进程 P2 有周期 p2 = 70 和 t2 = 25 .

(p 表示周期，t 表示 cpu 执行时间)

则（30/50 + 25/70）* 100% = 95.714% ,理论上这两个进程可以被调度。现分别使用单调速率调度和 EDF 分析之：

a. 单调速率调度

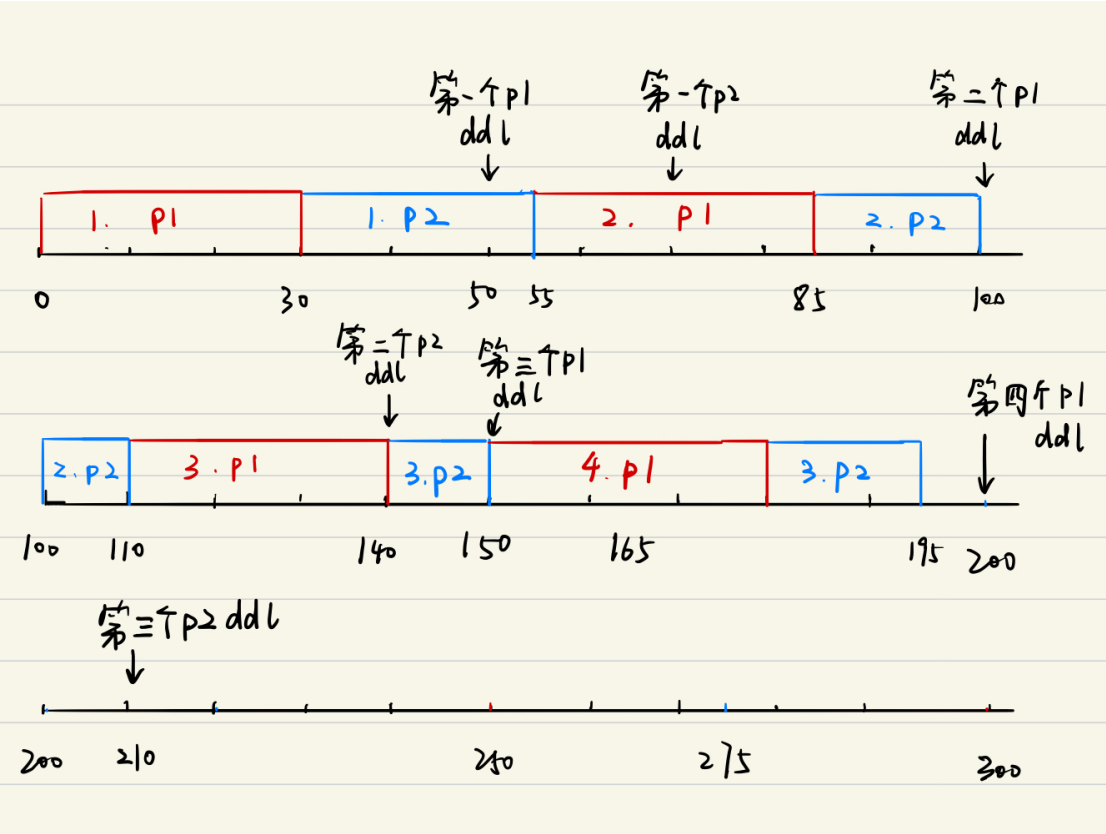
抢占+静态优先级，执行情况如下图：



可见，第一个 p2 进程完成于其截止期限之后（时间分别是 85、70），即此例中，单调速率调度 **无法满足** 与进程关联的截止期限要求

b. EDF

动态分配进程优先级，执行情况如下图：



注意到在时间为 150 时，第 4 周期 P1 的截止期限（200）比第 3 周期 P2 的截止期限（210）早，在 150 时，进程 P1 的优先级大于正在运行的 P2 的优先级，因此发生了一次切换

而在时间为 195 时，没有进程需要 cpu 执行，系统空闲直到 200，之后第 5 周期的 P1 进程再次被调度。因此可以说明，EDF 可以满足此例的 2 个进程的截止期限要求

所以对于此例中的 P1、P2，单速率截止调度在满足进程截止期限方面，比 EDF 要差