习题课讲解（第5-7章）

北京邮电大学 刘炳言

2022.12.12

第五章作业-作业1

例 1：假定要在一台处理器上执行如下图所示的作业，它们在 0 时刻以 1，2，3，4，5 的顺序到达。给出采用下列调度算法时的调度顺序、平均周转时间(turnaround time)和平均响应时间(response time)

（1）FCFS

（2）RR(时间片为 1，不考虑优先级)

（3）非抢占式 SJF(shortest job first)

（4）非抢占式优先级调度（数字小的优先级大）

作业	执行时间	优先级
1	10	3
2	1	1
3	2	2
4	3	4
5	5	2

Answer:

画出调度顺序

（1）FCFS: （2 分）

P1	P2	P3	P4	P5

0 10 11 13 16 21

平均响应时间＝（0＋10＋11＋13＋16）/ 5 ＝ 10

平均周转时间＝（10＋11＋13＋16＋21）/ 5 ＝ 14.2

（2） RR(TQ=1)

P1	P2	P3	P4	P5	P1	P3	P4	P5	P1	P4	P5	P1	P5	P1	P5	P1	P1	P1	P1	P1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
20 21

平均响应时间＝（0＋1＋2＋3＋4）/ 5 ＝ 2

平均周转时间＝（21＋2＋7＋11＋16）/ 5 ＝ 11.4

（3） SJF

P2	P3	P4	P5	P1

0　　1　　3　　6　　11　　21

平均响应时间＝（11＋0＋1＋3＋6）/ 5 ＝ 4.2

平均周转时间＝（21＋1＋3＋6＋11）/ 5 ＝ 8.4

（4） Priority（2 分）

P2	P3	P5	P1	P4

0　　1　　3　　8　　18　　21

平均响应时间＝（8＋0＋1＋18＋3）/ 5 ＝ 6

平均周转时间＝（18＋1＋3＋21＋8）/ 5 ＝ 10.2

第五章作业—作业2

- There are five tasks P_1-P_5 in a single processor system. Assuming that the arrival time and the burst time is shown in the table below. Assuming that all five tasks need only CPU and do not need I/O processing.
- 1) If the system uses HRRF to schedule, please draw a Gantt chart and calculate the average waiting time and the average turnaround time. （留学生班没有这一问）
- 2) If the system uses preemptive SJF schedule (if the shortest remaining job time is the same, using first come first serve), please draw a Gantt chart and calculate the average waiting time and the average turnaround time

process	Arrival time	Burst time
P_1	0	16
P_2	4	5
P_3	1	8
P_4	3	3
P_5	6	4

第五章作业—作业2

(1) .

调度顺序：1、4、5、2、3

进程	到达时间	开始时间	执行时间	结束时间	等待时间	周转时间
1	0	0	16	16	0	16
2	4	23	5	28	19	24
3	1	28	8	36	27	35
4	3	16	3	19	13	16
5	6	19	4	23	13	17

甘特图:

1	4	5	2	3

0　16　19　23　28　36

平均等待时间：(0 + 19 + 27 + 13 + 13) / 5 = 14.4

平均周转时间：(16 + 24 + 35 + 16 + 17) / 5 = 21.6

第五章作业—作业2

(2) .

甘特图:

1	3	4	5	2	3	1

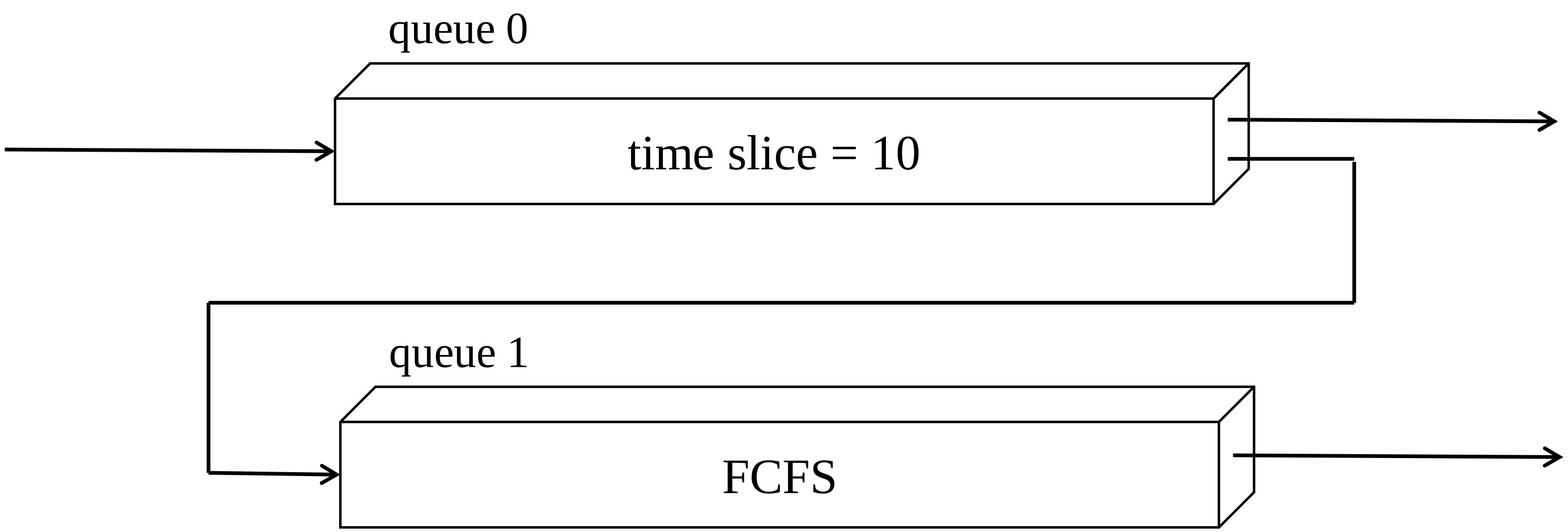
0 1 3 6 10 15 21 36

平均等待时间:(20 + 6 + 12 + 0 + 0) / 5 = 7.6

平均周转时间:(36 + 11 + 20 + 3 + 4) / 5 = 14.8

第五章作业—作业3

- As shown below, OS takes a two-level feedback-queue scheme to allocate CPU for concurrent processes. A process entering the system is at first put in queue 0, and sequentially given a CPU time slice of 10 milliseconds. If it does not finish within this time, it is moved to the tail of queue 1. Processes in queue 1 run on FCFS scheduling, but are permitted to run only when there is no process in queue 0. When a process P_i in queue 1 is running on CPU and a new process P_j enters the system, P_j will preempt the CPU occupied by P_i.



第五章作业—作业3

- Consider the processes P_0, P_1, P_2, P_3. For $0 \leq i \leq 3$, the arrival time, the length of the CPU burst time, and the priority of each P_i are given as below

Process	Arrival time	Burst Time	Priority
P_0	0.0	6.0	10
P_1	4.0	15.0	8
P_2	8.0	4.0	6
P_3	12.0	13.0	4

- For the snapshot shown above, suppose that two-level feedback-queue scheduling is employed
  - (1) Draw the Gantt chart that illustrates the execution of these processes.
  - (2) What are the turnaround times for the four processes?

第五章作业—作业3

(1) .

甘特图：

0	1	2	3	1	3

0　6　16　20　30　35　38

(2) .

进程	周转时间
P0	6
P1	31
P2	12
P3	26

第六章作业

- **1 老和尚-小和尚挑水**

- 考虑
  - 并发活动对象：和尚/并发进程
  - 对象业务逻辑：工作流程
  - 共享资源：需要互斥的共享资源

- Question (北京邮电大学计算机考研试题)
  - 某寺庙，小、老和尚若干
  - 水缸，由小和尚提水入缸(向缸中倒水),老和尚从缸中提水饮用，水缸可容10 桶水
  - 水取自同一井中，水井径窄，每次只能容一个桶取水
  - 水桶总数为3个。每次向缸中倒水、或从缸中取水仅为1 桶，且不可同时进行

试给出有关从缸中取水和向缸中倒水的算法描述

第六章作业

- **1 老和尚-小和尚挑水**

- 2类进程
  - 小和尚进程：获取水桶；用水桶从井中取水；向缸中倒水；归还水桶
  - 老和尚进程：获取水桶；用水桶从缸中取水；归还水桶
- 信号量设置
  - 互斥/临界资源：水井 (一次仅一个水桶进出)，设信号量mutexl
  - 互斥/临界资源：水缸 (一次倒入、取一桶水)，设信号量mutex2
  - 对水桶资源的互斥竞争使用：3个水桶无论从井中取水还是人出水缸都是一次一个，设信号量count，抢不到水桶的进程只好等待；
  - 水缸满时，不可倒水，设信号量empty 控制入水量
  - 水缸空时，不可出水，设信号量full，控制出水量

```
semaphore mutex1 = 1, mutex2 = 1;
semaphore empty = 10, full = 0;
semaphore count = 3;

SmallMonk(){
    while(1){
        wait(empty);        //水缸中还能装水
        wait(count);        //等水桶

        wait(mutex1);
        fetch_water();      //取水
        signal(mutex1);

        wait(mutex2);
        put_water();        //将水倒入水缸
        signal(mutex2);

        signal(count);
        signal(full);
    }
}
```

```

OldMonk(){
    while(1){
        wait(full);         //水缸里还有水
        wait(count);

        wait(mutex2);
        drink_water();      //从水缸中取水并饮用
        signal(mutex2);

        signal(count);
        signal(empty);
    }
}
```

第六章作业

- **2 多类型读-写者问题**【2个读者队列，1个写者队列】

- 假设有A、B、C三类用户（每类用户有多个）对数据库中的共享数据DATA进行并发读写，同步互斥要求如下
  - A类用户作为读者读DATA时，另外2类用户不允许访问DATA，但允许多个A类用户同时访问DATA；
  - 类似地，B类用户作为读者读DATA时，另外2类用户不允许访问DATA，但允许多个B类用户同时访问DATA；
  - 当1个C类用户作为写者修改DATA时，另外2类用户不允许访问DATA，同时也不允许其它C类用户访问DATA
- 采用信号量机制，描述A、B、C三类用户的行为

```
semaphore mutexA = 1, mutexB = 1;
semaphore rw = 1;
int countA = 0, countB = 0;

A(){
    while(1){
        wait(mutexA);
        if(countA == 0){
            wait(rw);       //争夺 data
        }
        countA++;
        signal(mutexA);
        read();
        wait(mutexA);
        countA--;
        if(countA == 0){
            signal(rw);
        }
        signal(mutexA);
    }
}
```

```
B(){
    while(1){
        wait(mutexB);
        if(countB == 0){
            wait(rw);       //争夺 data
        }
        countB++;
        signal(mutexB);
        read();
        wait(mutexB);
        countB--;
        if(countB == 0){
            signal(rw);
        }
        signal(mutexB);
    }
}

C(){
```

```
    while(1){
        wait(rw);           //争夺 data
        write();
        signal(rw);
    }
}
```

第六章作业

- **3牙科门诊（类似睡眠理发师问题）**
  - 校医院口腔科每天向患者提供20个挂号就诊名额
  - 患者到达医院后，
    - 如果有号，则挂号，在候诊室排队等待就医
    - 如果号已满，则离开医院
  - 三位医生在诊疗室为患者提供治疗服务，
    - 如果候诊室有患者并且诊疗室内有医生处于“休息”态，则从诊疗室挑选一位患者，安排一位医生为其治疗，医生转入“工作”状态；
    - 如果三位医生均处于“工作”态，候诊室内患者需等待
    - 当无患者候诊时，医生转入“休息”状态

第六章作业

- **3牙科门诊（类似睡眠理发师问题）**
  - 要求
    - 采用信号量机制，描述患者、医生的行为
    - 设置医生忙闲状态向量DState[3]，记录医生的“工作”、“休息”状态
    - 设置患者就诊状态向量PState[20]，记录挂号成功后的患者的“候诊”、“就医”状态
  - 注意：
    - 3个医生对共享数据结构DState[3]的修改必须互斥进行，引入互斥变量1，以控制医生间的互斥访问行为；
    - 20个患者对共享数据结构PState[20]的修改必须互斥进行，引入互斥变量2，以控制患者间的互斥访问行为；

```
semaphore doctors = 0, patients = 0;
semaphore Dmutex = 1, Pmutex = 1;
int countP = 0;
semaphore mutexC = 1;

enum{
    WORK, REST
} DState[3];

enum{
    WAIT, VISIT
} PState[20];

// 初始化 DState 和PState
for (int i = 0; i < 3; i++){
    DState[i] = REST;
}

for (int i = 0; i < 20; i++){
    PState[i] = WAIT;
}
Doctor(){
    while (1){
        int myNum = get_myNum(); //返回1 or 2 or 3
        wait(Dmutex);            //修改状态
        DState[myNum] = REST;
        signal(Dmutex);

        signal(doctors);
        wait(patients);          //有病人则进入忙碌状态

        wait(Dmutex);
        DState[myNum] = BUSY;
        signal(Dmutex);

        examineThePatient();     //进行检查
    }
}

Patient(){
    while(1){
        int myCount;
        wait(mutexC);
        if(countP < 20){         //判断是否还有号
            myCount = countP;    //取号
            countP++;
            signal(mutexC);
            signal(patients);    //进入病人队列
            wait(doctors);       //等待空闲医生

            wait(Pmutex);   //修改自己的状态
            PState[myCount] = VISIT;
            signal(Pmutex);
            goToADoctor();
        }
        else {
            signal(mutexC);
        }
    }
}
```

第七章作业-补充作业1

- There are 5 processes P_0, P_1, P_2, P_3, P_4; 3 resource types A with 10 instances (e.g. memory), B with 5 instances (disk), and C with 7 instances (tapes)
  - snapshot at time T_0:

	Allocation	*Max*	*Available*	*Need=MAX —Allocation*
	A B C	*A B C*	*A B C*	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

total resources : (10 5 7)

Can request for (3, 3, 0) by P_4 be granted?
Can request for (0, 2, 0) by P_0 be granted?

Can request for (3, 3, 0) by P_4 be granted?

- decide whether or not the P_4's request(3, 3, 0) can be granted
  - *request* (3, 3, 0) < $Need_4$ = (4, 3, 1)
  - check that *Request* ≤ *Available* (that is, (3, 3, 0) ≤ (3, 3, 2) ⇒ *true*), (3 3 0) are allocated to P_4

	Allocation	*Max*	*Available*	*Need=MAX —Allocation*
	A B C	*A B C*	*A B C*	A B C
P_0	0 1 0	7 5 3	0 0 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	3 3 2	4 3 3		1 0 1

- executing safety algorithm, no sequence satisfied safety requirement. So (3, 3, 0) requested by P4 cannot be granted.

Can request for (0, 2, 0) by P_0 be granted?

- decide whether or not the P_0's request(0, 2, 0) can be granted
  - *request* (0, 2, 0) < $Need_0$ = (7, 4, 3)
  - check that *Request* ≤ *Available* (that is, (0, 2, 0) ≤ (3, 3, 2) ⇒ *true*), (0 2 0) are allocated to P_0, and system enters a new safety state

	Allocation	*Max*	*Available*	*Need=MAX —Allocation*
	A B C	*A B C*	*A B C*	A B C
P_0	0 3 0	7 5 3	3 1 2	7 2 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

Can request for (0, 2, 0) by P_0 be granted?

- apply safety algorithm, for sequence $< P_3, P_1, P_2, P_0, P_4 >$

	Need: *A B C*	*Work*
step1. P_3	0 1 1	< (3 1 2)
step2. P_1	1 2 2	< (3 1 2) + Allocat3(2 1 1) =(5 2 3)
Step3. P_2	6 0 0	< (5 2 3) + Allocat1(2 0 0) =(7 2 3)
step4. P_0	7 2 3	< (7 2 3) + Allocat2(3 0 2)=(10 2 5)
Step5. P_4	4 3 1	< (10 2 5) + Allocat0 (0 3 0)=(10 5 5)

- ***In the current situation,*** the system is in a safe state since the sequence $< P_3, P_1, P_2, P_0, P_4 >$ satisfies safety criteria.

第七章作业-补充作业2

- Five processes P_0 through P_4; three resource types
  A with 6 instances, B *with* 3 instances, and C with 6 instances
- Given the snapshot at time T_0:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	1 0 1
P_1	2 0 1	2 0 2	
P_2	1 1 1	0 0 2	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

第七章作业-补充作业2

- Answer the following questions on the basis of deadlock-detecting algorithm
  - is the system in a deadlock-free state?   and why?
  - if P_2 requests two additional instance of type A, is there a deadlock in the system, and why?

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	1 0 1
P_1	2 0 1	2 0 2	
P_2	1 1 1	2 0 2	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Is the system in a deadlock-free state? and why?

- apply safety algorithm, for sequence $< \boldsymbol{P_0, P_3, P_1, P_2, P_4} >$

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	1 0 1
P_1	2 0 1	2 0 2	
P_2	1 1 1	0 0 2	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

In the current situation, the system is in a deadlock-free state since the sequence $< \boldsymbol{P_0, P_3, P_1, P_2, P_4} >$ satisfies safety criteria.

If P_2 requests two additional instance of type A, is there a deadlock in the system, and why?

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	1 0 1
P_1	2 0 1	2 0 2	
P_2	1 1 1	**2** 0 2	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

In the current situation, the system is in a deadlock-free state since the sequence **< P_0, P_3, P_1, P_2, P_4 >** satisfies safety criteria.

谢谢！
Q&A