《操作系统》课下作业 (OS-HW4)

中国人民大学 信息学院 崔冠宇 2018202147

P332, 6.5 Given the following state of a system:

The system comprises of five processes and four resources. P1–P5 denotes the set of processes. R1–R4 denotes the set of resources.

Total Existing Resources:

R1	R2	R3	R4
6	3	4	3

Snapshot at the initial time stage:

	Allocation				Claim			
	R1	R2	R3	R4	R1	R2	R3	R4
P1	3	0	1	1	6	2	1	1
P2	0	1	0	0	0	2	1	2
P3	1	1	1	0	3	2	1	0
P4	1	1	0	1	1	1	1	1
P5	0	0	0	0	2	1	1	1

a. Compute the Available vector.

b. Compute the Need Matrix.

c. Is the current allocation state safe? If so, give a safe sequence of the process. In addition, show how the Available (working array) changes as each process terminates.

d. If the request (1, 1, 0, 0) from P1 arrives, will it be correct to grant the request? Justify your decision.

解.

a. 由于 $\text{Available}_i = \text{Resource}_i - \sum_j \text{Allocation}_{ij}$, 故 $R_1 = 6-(3+0+1+1+0) = 1, R_2 = 3-(0+1+1+1+0) = 0, R_3 = 4-(1+0+1+0+0) = 2, R_4 = 3-(1+0+0+1+0) = 1$, 所以可用资源向量: $\text{Available} = (1,0,2,1)$.

b. 由于 Need = Claim − Allocation, 故需求矩阵:

$$\text{Need} = \begin{pmatrix} 3 & 2 & 0 & 0 \\ 0 & 1 & 1 & 2 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 1 & 1 & 1 \end{pmatrix}.$$

c. 应用银行家算法, 得到一个安全进程序列(不唯一): 初始——P4——P2——P3——P1——P5, 对应的进程结束时的可用资源向量序列为: $(1,0,2,1) \to (2,1,2,2) \to (2,2,2,2) \to (3,3,3,2) \to (6,3,4,3) \to (6,3,4,3)$.

d. 假定接受请求, P1 的 Allocation 向量将变为 (4,1,1,1), Available 向量为 (0,-1,2,1), 不合法, 所以拒绝请求, 阻塞进程.

P333, 6.6 In the code below, three processes are competing for six resources labeled A to F.

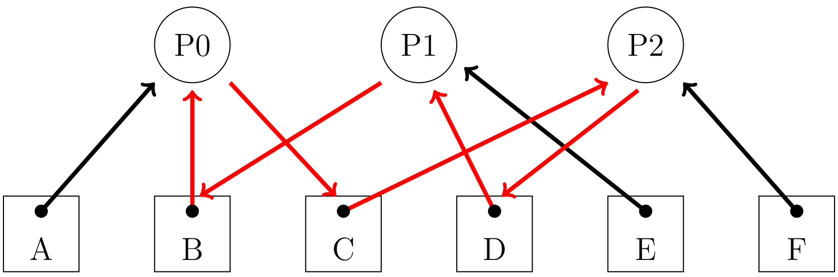
a. Using a resource allocation graph(see Figures 6.5 and 6.6), show the possibility of a deadlock in this implementation.

b. Modify the order of some of the get requests to prevent the possibility of any deadlock. You cannot move requests across procedures, only change the order inside each procedure. Use a resource allocation graph to justify your answer.

`void P0()`
`{`
` while (true) {`
` get(A);`
` get(B);`
` get(C);`
` // critical region:`
` // use A, B, C`
` release(A);`
` release(B);`
` release(C);`
` }`
`}`
	`void P1()`
	`{`
	` while (true) {`
	` get(D);`
	` get(E);`
	` get(B);`
	` // critical region:`
	` // use D, E, B`
	` release(D);`
	` release(E);`
	` release(B);`
	` }`
	`}`
		`void P2()`
		`{`
		` while (true) {`
		` get(C);`
		` get(F);`
		` get(D);`
		` // critical region:`
		` // use C, F, D`
		` release(C);`
		` release(F);`
		` release(D);`
		` }`
		`}`

解.

a. 如下图所示:



当 P0 占有 A、B, 请求 C; P1 占有 D、E, 请求 B; P2 占有 C、F, 请求 D 时, 出现了循环等待(图中红箭头), 发生死锁.

b. 只要改变进程中资源请求的顺序, 打破之前出现的循环请求即可, 例子: P0: A, B, C; P1: B, D, E; P2: C, D, F.

P333, 6.7 A spooling system (see Figure 6.17) consists of an input process I, a user process P, and an output process O connected by two buffers. The processes exchange data in blocks of equal size.

These blocks are buffered on a disk using a floating boundary between the input and the output buffers, depending on the speed of the processes.

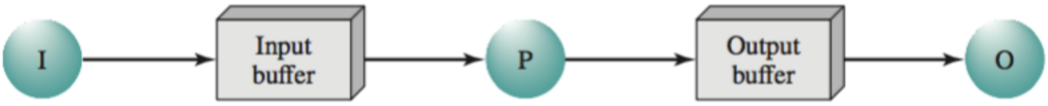


Figure 6.17 A Spooling System

The communication primitives used ensure that the following resource constraint is satisfied:

$$i + o \leq \max$$

where

max = maximum number of blocks on disk

i = number of input blocks on disk

o = number of output blocks on disk

The following is known about the processes:

1. As long as the environment supplies data, process I will eventually input it to the disk (provided disk space becomes available).

2. As long as input is available on the disk, process P will eventually consume it and output a finite amount of data on the disk for each block input (provided disk space becomes available).

3. As long as output is available on the disk, process O will eventually consume it.

Show that this system can become deadlocked.

解.

当进程 I 速度极快, 已经将输入写满, 即 $i = \max$ 时; 与此同时, 进程 P 等待将输入转换为输出写入磁盘, 但是磁盘没有可用位置; 进程 O 因为没有输出内容而等待. 这样就产生了死锁.

P335, 6.14 Suppose the following two processes, foo and bar, are executed concurrently and share the semaphore variables S and R (each initialized to 1) and the integer variable x (initialized to 0).

<pre>void foo( ) {
   do {
     semWait(S);
     semWait(R);
     x++;
     semSignal(S);
     SemSignal(R);
  } while (1);
}</pre>
	<pre>void bar( ) {
	do {
	     semWait(R);
	     semWait(S);
	     x--;
	     semSignal(S;
	     SemSignal(R);
	} while (1);
	}</pre>

a. Can the concurrent execution of these two processes result in one or both being blocked forever? If yes, give an execution sequence in which one or both are blocked forever.

b. Can the concurrent execution of these two processes result in the indefinite postponement of one of them? If yes, give an execution sequence in which one is indefinitely postponed.

解.

a. 会. 例如 foo 执行 `semWait(S)`; bar 执行 `semWait(R)`, 均获得了信号量. 然而此时继续运行, foo 执行 `semWait(R)`, 被阻塞; bar 执行 `semWait(S)`, 也被阻塞, 于是出现了循环等待, 二者就因为等不到对方的 `semSignal()` 就都被永久阻塞.

b. 不会. 如果一个进程被 `semWait()` 阻塞, 则有下面两种情况:

1. 另一个进程就如(a)所描述的, 也处于阻塞态, 陷入死锁;
2. 另一个进程已经进入临界区, 离开临界区后会执行 `semSignal()` 释放信号量, 这就唤醒了被阻塞的进程.

所以要么出现死锁(一个进程占有一个资源), 要么顺利完成一次循环(一个进程占有全部资源), 不会出现一个进程被无限期延后的情况.