# Homework 3

[Synchronization+Deadlock]

1. Two processes enter into critical zone by using semaphores ***mutex***, at first ***mutex*** = 1, when ***mutex*** = 0 means **\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

A）none process has entered the critical zone

B）one process has entered the critical zone

C）two process have entered the critical zone

D）one process has entered the critical zone, another one is waiting

1. There are 4 processes sharing 18 resources. For dynamic avoid deadlock, the most number of resource which every process could request is （ ）

A）4 B）5

C）6 D）7

1. ④\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is a kind of special variable which can only be modified by two operations: P() and V(). It can be used to implement the control mechanism among several asynchronously concurrent running processes, including ⑤\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and ⑥\_\_\_\_\_\_\_\_\_\_\_\_\_. ⑤ means to share resources exclusively, while ⑥ applies to control the execution order between two processes in logic. ⑦\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is also a special data structure used for the above control mechanism, which combines ⑤ and ⑥ together in one module. The ⑧\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_in ⑦ is used to implement the function of ⑥.

④⑤⑥⑦⑧： scheduling, class, process, mutual exclusion, semaphore, control variable, condition variable, monitor, synchronization, shared variable, procedure, dispatching

1. A \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is a kind of special variable which can only be modified by two operations: P() and V(). It was proposed by Dr. \_\_\_\_\_\_\_\_\_\_\_in 1965, and there are mainly two types \_\_\_\_\_\_\_\_\_\_\_\_\_\_ and\_\_\_\_\_\_\_\_\_\_\_. can be used to support the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ constraint by initializing its value as 1, and using paired P() and V() around the \_\_\_\_\_\_\_\_\_\_.

：scheduler, class, process, mutual exclusion, semaphore, control variable, binary semaphore, condition variable, monitor, synchronization, counting semaphore, scheduling, shared variable, procedure, dispatching, critical section, race condition

: Alan M. Turing, Edsger W. Dijkstra, John Vincent Atanasoff, Donald E. Knuth, Nicolas Worth

1. The two essential conditions which may cause the deadlock are ⑨\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and ⑩\_\_\_\_\_\_\_\_\_\_\_\_\_. The four necessary conditions of the deadlock are Mutual exclusion, ⑪\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, no-preemption and ⑫\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. The Banker’s algorithm is used to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ the deadlock.

⑨: inappropriate allocation of the shared resources, inadequate resources, improper scheduling of the jobs, too many users

⑩: too many processes, too busy scheduling of the processes, CPU is too slow, the execution order of related processes are not well designed

⑪: hold and block, hold and release, hold and wait, release and block

⑫: circular, circular wait, resource numbering, unordered resources

⑬: avoid, prevent, control, simulate, unchain

1. The critical section problem is to design a protocol that the processes can use so that their action will not depend on the order in which their execution is interleaved (possibly on many processors). There are 3 criteria for the critical section problem solution, namely ⑰\_\_\_\_\_\_\_\_\_\_\_\_\_\_, ⑱ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, ⑲\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. For any solution, we should check all these three criteria. If any one of them is not satisfied, the solution cannot be correct. Among the solutions, semaphore-based mechanism is the most popular now, in which the semaphore concept was first proposed by ⑳\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ in 1965.

⑰⑱⑲: no-preemption, mutual exclusion, circular wait, hold and wait, progress, bounded waiting,

⑳: Leslie Lamport, Alan Mathison Turing, Edsger W. Dijkstra, John Vincent Atanasoff, Donald E. Knuth

1. Two processes, A and B, each need three records, 1, 2, 3, in a database. If A asks for them in the order 1, 2, 3, then in which order B asks for them, deadlock is not possible （ ）

A）1, 2, 3 B）2, 1, 3

C）3, 2, 1 D）1, 3, 2

1. When using a counting semaphore to control the usage of 5 printers shared among many processes, which value could not occur for the counting semaphore?

A. -6 B.-5 C. 5 D. 6

1. Which of the following statement about resource-allocation graph and deadlock is wrong?

A. If graph contains no cycles then there has no deadlock

B. If graph contains a cycle and only one instance per resource type, then there must be deadlock

C. If there is a deadlock, then the graph must contain a cycle.

D. If there are no deadlocks, then the graph must not contain any cycle.

1. Figure 1 illustrates a bridge, and the arrows show the directions of the corresponding cars. Only one car is allowed on the bridge at any time, but several cars are allowed to pass the bridge one by one if they are for the same direction. You are required to fill the blanks in following code which is used to cope with this synchronization problem using P and V operations.

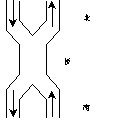


Figure : A bridge (Direction for top is north)

Var integer mutex ，availn ，avails;

// availn (for North) and avails (for South) are semaphores to synchronize the passing cars

availn = m; // **when it’s m, it means only the cars from this direction could pass the bridge**

avails =0; // **when it’s 0, it means only the cars from this direction are forbidden to pass the bridge**

mutex = 1; // mutual exclusion for accessing the bridge

**COBEGIN**

**Car for South**:

BEGIN

P(\_\_\_\_\_\_\_\_\_\_\_\_);

P(\_\_\_\_\_\_\_\_\_\_\_\_);

Cross the bridge;

V(\_\_\_\_\_\_\_\_\_\_\_);

V(\_\_\_\_\_\_\_\_\_\_);

END;

**Car for North**:

BEGIN

P(\_\_\_\_\_\_\_\_\_\_);

P(\_\_\_\_\_\_\_\_\_);

Cross the bridge;

V(\_\_\_\_\_\_\_\_);

V(\_\_\_\_\_\_\_\_\_);

END;

**COEND**;

1. Assume there are 3 kinds of resources (A B C) and 5 processes (P1, P2, P3, P4, P5) in a system. The amounts of resource A are 17, resource B are 5, resource C are 20. The system state is as follows table at time T0. Please avoid deadlock by using Bank’s algorithm.

|  |  |  |  |
| --- | --- | --- | --- |
| process | Max used | allocated | left |
| A B C | A B C | A B C |
| P1 | 5 5 9 | 2 1 2 | 2 3 3 |
| P2 | 5 3 6 | 4 0 2 |  |
| P3 | 4 0 11 | 4 0 5 |  |
| P4 | 4 2 5 | 2 0 4 |  |
| P5 | 4 2 4 | 3 1 4 |  |

Please Answer:

1. Is it safety when it is in time T0? If yes, write down the safety order; if not, write down why?
2. If P2 now requests resource (0, 3, 4) at time T0, could you allocate them? If yes, write down the new safety order; if not, write down why?
3. If now P4 requests resource (2, 0, 1) after (b), could you allocate them? If yes, write down the new safety order; if not, write down why?
4. If now P1 requests resource (0, 2, 0) after (c), could you allocate them? If yes, write down the new safety order; if not, write down why?
5. There is an empty plate on the table. Assume there are two kinds of fruits, the apple and the orange. Every time, only one fruit could be put on the plate. There are three persons – a waiter, a male customer and a female customer – who share the plate. The waiter can put an apple or an orange on the plate one at a time. The male customer is waiting to eat an apple, while the female customer is waiting to eat an orange. The following program is a solution to the problem, please fill the blank.

begin

empty, full1, full2:semaphore;

empty:= 1 ;

full1= 0 ;

full2:= 0 ;;

cobegin

processor waiter

begin

wait（ ）;

if put an apple

signal（ ）;

else

signal（full2）;

end

processor male customer

begin

wait（ full1 ） ;

get an apple to eat;

signal（ ）;

end

processor female customer

begin

wait（ ） ;

get an orange to eat;

signal（ ）;

end

coend;

end

1. Readers-Writers Problem is a quite popular synchronization problem in OS. One of its typical sub-problems is the First Readers-Writers Problem, in which readers have priority over writers. That is, unless a writer has permission to access the object, any reader requesting access to the object will get it. (**Note this may result in a writer waiting indefinitely to access the object, AKA starvation.**) Following is an unfinished pseudo code for this problem, please complete it. (5 scores)

**Begin**

readcountmutex, wmutex: semaphore; // 两个互斥信号量

readcount: Integer; // 面向Reader的计数器

readcountmutex = wmutex = 1;

rcount = 0;

**Cobegin**

**Process procedure Reader**

**begin**

**repeat**

// *do something*

P(**readcountmutex**);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

if (readcount ==1) then P(**wmutex**);

V(**readcountmutex**);

***perform read operations***;

P(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

readcount:= readcount - 1;

if (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) then V(\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

V(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

// *do something else*

**until** false;

**end**

**Process procedure Writer**

**begin**

**repeat**

P(**wmutex**);

***perform write operations***;

V(**wmutex**);

**until** false

**end**

**CoEnd**

**End**