Semaphore: Exercise

1. The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initialized as S0 = 1, S1 = 0, S2 = 0.

|  |  |  |
| --- | --- | --- |
| Process P0 | Process P1 | Process P2 |
| while(true) { P(S0); print ‘0’; V(S1); V(S2); } | P(S1); V(S0); | P(S2); V(S0); |

How many times will P0 print ‘0’?  
a) At least twice  
b) Exactly twice  
c) Exactly thrice  
d) Exactly once

Answer: a

1. Each process Pi, i = 0,1,2,3,……,9 is coded as follows :  
   repeat  
   P(mutex)  
   {

Critical Section

}  
V(mutex)  
forever

The code for P10 is identical except that it uses V(mutex) instead of P(mutex). What is the largest number of processes that can be inside the critical section at any moment (the mutex being initialized to 1)?  
a) 1  
b) 2  
c) 3  
d) None of these

Answer: c  
Explanation: Any one of the 9 processes can get into critical section after executing P(mutex) which decrements the mutex value to 0. At this time P10 can enter critical section by incrementing the value to 1. Now any of the 9 processes can enter the critical section by again decrementing the mutex value to 0. None of the remaining processes can get into their critical sections.

1. Two processes, P1 and P2, need to access a critical section of code. Consider the following synchronization construct used by the processes:

|  |  |
| --- | --- |
| Process P1 | Process P2 |
| while(true) { w1 = true; while (w2 == true); Critical section w1 = false; } Remainder Section | while(true) { w2 = true; while (w1 == true); Critical section w2 = false; } Remainder Section |

Here, w1 and w2 are shared variables, which are initialized to false. Which one of the following statements is TRUE about the above construct?  
a) It does not ensure mutual exclusion  
b) It does not ensure bounded waiting  
c) It requires that processes enter the critical section in strict alternation  
d) It does not prevent deadlocks, but ensures mutual exclusion

Answer: d

1. Suppose we want to synchronize two concurrent processes P and Q using binary semaphores S and T. The code for the processes P and Q is shown below.

|  |  |
| --- | --- |
| Process P: | Process Q: |
| while (1) {  W:  print '0';  print '0';  X:  } | while (1) {  Y:  print '1';  print '1';  Z:  } |

Synchronization statements can be inserted only at points W, X, Y and Z.  
 ***Which of the following will always lead to an output staring with ‘001100110011’ ?***  
**(A)** P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S and T initially 1  
**(B)** P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S initially 1, and T initially 0  
**(C)** P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S and T initially 1  
**(D)** P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S initially 1, and T initially 0  
  
  
**Answer:** **(B)**   
  
**Explanation:** P(S) means wait on semaphore ‘S’ and V(S) means signal on semaphore ‘S’.

Initially, we assume S = 1 and T = 0 to support mutual exclusion in process P and Q.  
Since S = 1, only process P will be executed and wait(S) will decrement the value of S. Therefore, S = 0.  
At the same instant, in process Q, value of T = 0. Therefore, in process Q, control will be stuck in while loop till the time process P prints 00 and increments the value of T by calling the function V(T).  
While the control is in process Q, semaphore S = 0 and process P would be stuck in while loop and would not execute till the time process Q prints 11 and makes the value of S = 1 by calling the function V(S).  
This whole process will repeat to give the output 00 11 00 11 … .

1. Bill and Ben work in a cafe that specialises in a soup and sandwich combination. Bill makes two rounds of sandwiches while, at the same time, Ben makes two bowls of soup. When the sandwiches and soup are ready Bill and Ben each, at the same time, serve a soup and sandwich combination to a customer. These actions are repeated throughout the day. [Note: Bill cannot serve until Ben has made the soup and Ben cannot serve until Bill has made the sandwiches.] Let Bill and Ben be represented by two processes with the following structure:

**Variable declarations**

**process Bill {**

**while (true) {**

**// Bill makes sandwiches //**

**SYNCH 1;**

**// Bill delivers //**

**}**

**}**

**process Ben {**

**while (true) {**

**// Ben makes soup //**

**SYNCH 2;**

**// Ben delivers //**

**}**

**}**

Provide variable declarations (including any initialization) and code for SYNCH 1 and SYNCH 2 using semaphores so that the above algorithm behaves correctly.

Solution

**Semaphore first = 0;**

**Semaphore second = 0;**

**process Bill {**

**while (true) {**

**// Bill makes sandwiches //**

**down(first);**

**up(second);**

**// Bill delivers //**

**}**

**}**

**process Ben {**

**while (true) {**

**// Ben makes soup //**

**up(first);**

**down(second);**

**// Ben delivers //**

**}**

**}**

1. The following pair of processes share a common variable X:

Process A Process B

int Y; int Z;

A1: Y = X\*2; B1: Z = X+1;

A2: X = Y; B2: X = Z;

X is set to 5 before either process begins execution. As usual, statements within a process are executed sequentially, but statements in process A may execute in any order with respect to statements in process B.

How many different values of X are possible after both processes finish executing?

There are four possible values for X. Here are the possible ways in which statements from A and B can be interleaved.

A1 A2 B1 B2: X = 11

A1 B1 A2 B2: X = 6

A1 B1 B2 A2: X = 10

B1 A1 B2 A2: X = 10

B1 A1 A2 B2: X = 6

B1 B2 A1 A2: X = 12

Suppose the programs are modified as follows to use a shared binary semaphore S:

Process A Process B

int Y; int Z;

wait(S); wait(S);

A1: Y = X\*2; B1: Z = X+1;

A2: X = Y; B2: X = Z;

signal(S); signal(S);

S is set to 1 before either process begins execution and, as before, X is set to 5.

Now, how many different values of X are possible after both processes finish executing?

The semaphore S ensures that, once begun, the statements from either process execute without interrupts. So now the possible ways in which statements from A and B can be interleaved are:

A1 A2 B1 B2: X = 11

B1 B2 A1 A2: X = 12

Finally, suppose the programs are modified as follows to use a shared binary semaphore T:

Process A Process B

int Y; int Z;

A1: Y = X\*2; B1: wait(T);

A2: X = Y; B2: Z = X+1;

signal(T); X = Z;

T is set to 0 before either process begins execution and, as before, X is set to 5. Now, how many different values of X are possible after both processes finish executing?

The semaphore T ensures that all the statements from A finish execution before B begins. So now there is only one way in which statements from A and B can be interleaved:

A1 A2 B1 B2: X = 11

1. The following pair of processes share a common set of variables: "counter", "tempA" and "tempB":

Process A Process B

... ...

A1: tempA = counter + 1; B1: tempB = counter + 2;

A2: counter = tempA; B2: counter = tempB;

... ...

The variable "counter" initially has the value 10 before either process begins to execute.

What different values of "counter" are possible when both processes have finished executing? Give an order of execution of statements from processes A and B that would yield each of the values you give. For example, execution order A1, A2, B1, B2 would yield the value 13.

There are three possible values for X. Here are the possible ways in which statements from A and B can be interleaved.

A1 A2 B1 B2: X = 13

A1 B1 A2 B2: X = 12

A1 B1 B2 A2: X = 11

B1 A1 B2 A2: X = 11

B1 A1 A2 B2: X = 12

B1 B2 A1 A2: X = 13

Modify the above programs for processes A and B by adding appropriate signal and wait operations on the binary semaphore "sync" such that the only possible final value of "counter" is 13. Indicate what should be the initial value of the semaphore "sync".

We need to ensure that A and B run uniterrupted, but it doesn't matter which runs first.

semaphore sync = 1;

Process A Process B

wait(sync); wait(sync);

A1: tempA = counter + 1; B1: tempB = counter + 2;

A2: counter = tempA; B2: counter = tempB;

signal(sync); signal(sync);

Draw a precedence graph that describes all the possible orderings of executions of statements A1, A2, B1 and B2 that yield the a final value of 11 for "counter".

A1 B1

\ /

B2

|

A2

Modify the original programs for processes A and B by adding binary semaphores and signal and wait operations to guarantee that the final result of executing the two processes will be "counter" = 11. Give the initial values for every semaphore you introduce. Try to put the minimum number of constraints on the ordering of statements. In other words, don't just pick one ordering that will yield 11 and enforce that one by means of semaphores; instead, enforce only the essential precedence constraints marked in your solution to question 3.

semaphore s1 = 0;

semaphore s2 = 0;

Process A Process B

A1: tempA = counter + 1; B1: tempB = counter + 2;

signal(s1); wait(s1);

wait(s2); B2: counter = tempB;

A2: counter = tempA; signal(s2);