1. Provide an example using the producer-consumer problem that **illustrates** why atomic variables are not enough for synchronization for all circumstances.   
   **Note:** Your answer should include which variable to be atomic and in which part the use of atomic variable is not enough for synchronization.

Answer:

Producer:

while (true) {/\* produce an item in next produced \*/

while (count==BUFFER\_SIZE);

/\* do nothing \*/

buffer[in] = char\_c;

in = (in + 1)% BUFFER\_SIZE;

counter++;

}

Consumer:

while (true) {

while (count == 0)

; /\* do nothing \*/

char\_c= buffer[out];

out = (out+1)% BUFFER\_SIZE;

counter--;

/\* consume the item in next consumed \*/

}

The variable which be atomic is count

Synchronization problem in buffer in , out in char c we wanted to add lock here

====================================================

1. On modern multi-core computing systems *spinlocks* are widely used in many Operating systems. What is the advantage that spinlocks provide? And When this advantage applies?

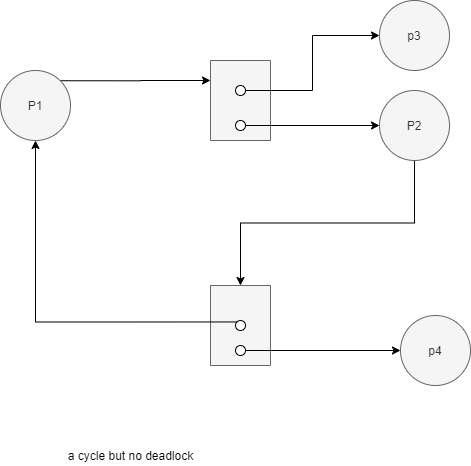
Answer:

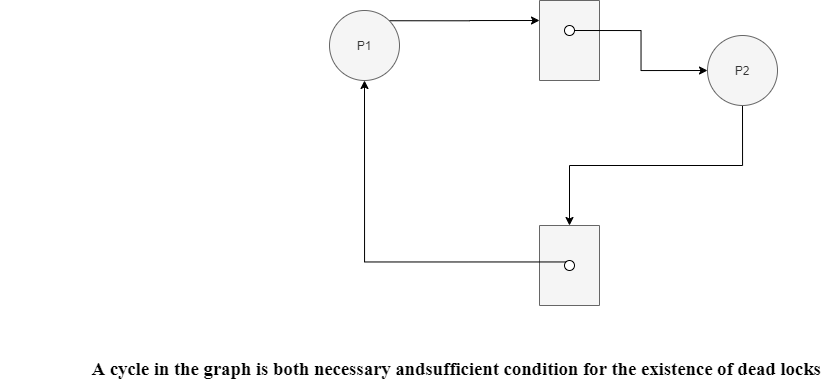
-SpinLock performs busy waiting and can offer better performance when used in multi-core systems especially and it is typically used when working with interrupts to perform busy waiting inside a loop till the resource is made available. SpinLock don't cause the thread to be preempted, rather, it continues to spin till lock on the resource is released.

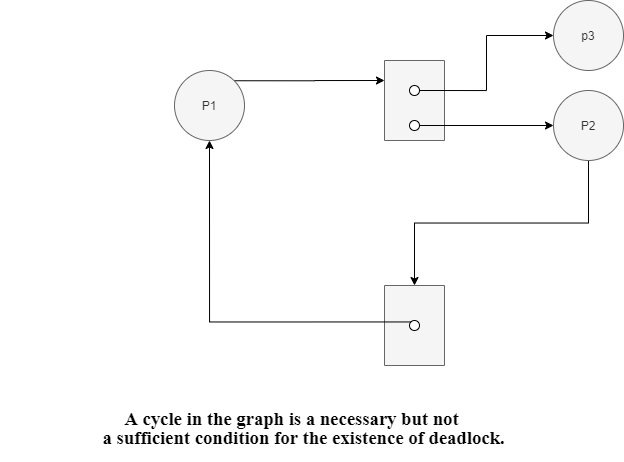
-use it only when the lock hold-times of a reasonably small duration.

1. Draw a resource allocation graph that shows:
   1. A cycle in the graph is both necessary and sufficient condition for the existence of dead locks.
   2. A cycle in the graph is a necessary but not a sufficient condition for the existence of deadlock.
   3. A cycle exists. However, there is no deadlock

Answer:







----------------------------------------------------------------

1. Identify how each of the following entities deals with a deadlock.   
   **Note** that we can generally deal with deadlocks by ignoring the problem, preventing/avoiding it, or by detecting it then recover from it.
   1. Linux based operating systems
   2. Kernel developer
   3. Application developer
   4. Databases

Answer:

1. ignoring
2. detecting
3. preventing
4. detecting
5. In the following code, three processes produce output using the routine printf and synchronize using two semaphores L and R.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Semaphore L is initialized to 3 and semaphore R is initialized to 0.

* 1. What is the smallest number of A's that might be printed when this set of processes runs?
  2. Is CABABDDCABCABD a possible output sequence when this set of processes runs?
  3. Is CABACDBCABDD a possible output sequence when this set of processes runs?

**Answer:**

1. If process 3 runs after process 1

The process 2 never executes and the smallest number is 0

1. No ,

Start : L=3,R=0

When

Print C : L=2,R=1

Print: A: L=2,R=0

Print B : L=2 ,R=1

Print: A: L=2,R=0

Print B : L=2 ,R=1

Print D : L=2,R=0

Print D -> it is impossible as R=0

1. Yes ,

When

Print C : L=2,R=1

Print: A: L=2,R=0

Print B : L=2 ,R=1

Print: A: L=2,R=0

Print C : L=1 ,R=1

Print D : L=1,R=0

Print B:L=1,R=1

print C:L=0 ,R=2

print A:L=0,R=1

print B L=0, R=2

print D:L=0 ,R=1

print D:L=0 ,R=0

1. Consider the following snapshot of a system:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Allocation | | | | Max | | | |
| A | B | C | D | A | B | C | D |
|  | 1 | 2 | 0 | 2 | 4 | 3 | 1 | 6 |
|  | 0 | 1 | 1 | 2 | 2 | 4 | 2 | 4 |
|  | 1 | 2 | 4 | 0 | 3 | 6 | 5 | 1 |
|  | 1 | 2 | 0 | 1 | 2 | 6 | 2 | 3 |
|  | 1 | 0 | 0 | 1 | 3 | 1 | 1 | 2 |

Using the **banker’s algorithm**, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the threads may complete. Otherwise, illustrate why the state is unsafe.



**Answer:**

**Needs**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| T0 | 3 | 1 | 1 | 4 |
| T1 | 2 | 3 | 1 | 2 |
| T2 | 2 | 4 | 1 | 1 |
| T3 | 1 | 4 | 2 | 2 |
| T4 | 2 | 1 | 1 | 1 |



It is safe

1-t4 available ( 3,2,2,4)

2- t0 available(4,4,2,6)

3-t1 available(4,5,3,7)

4- t2 available(5,7,7,7)

5- t3 available (6,8,7,8)

-----------------------------------------------

b- It is safe

1. t2 available(5,6,5,1)
2. t4 available(6,6,5,1)
3. t3 available(7,8,5,3)
4. t1 available(7,9,6,5)
5. t0 available(8,11,6,7)

c-no process complete as the available of instant of B =0 and all process need instant of B

d-it is safe

1-t3 available(2,7,2,3)

2-t1 available(2,8,3,5)

3-t2 available(3,10,7,5)

4-t4 available(4,10,7,6)

5-t0 available(5,12,7,8)

1. Consider the following snapshot of a system:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Allocation | | | | Max | | | |
| A | B | C | D | A | B | C | D |
|  | 3 | 1 | 4 | 1 | 6 | 4 | 7 | 3 |
|  | 2 | 1 | 0 | 2 | 4 | 2 | 3 | 2 |
|  | 2 | 4 | 1 | 3 | 2 | 5 | 3 | 3 |
|  | 4 | 1 | 1 | 0 | 6 | 3 | 3 | 2 |
|  | 2 | 2 | 2 | 1 | 5 | 6 | 7 | 5 |

Available =

Answer the following questions using the banker’s algorithm:

* 1. Illustrate that the system is in a safe state by demonstrating an order in which the threads may complete.
  2. If a request from thread arrives for , can the request be granted immediately?
  3. If a request from thread arrives for , can the request be granted immediately?
  4. If a request from thread arrives for , can the request be granted immediately?

**Answer :**

1. **T2,T0,T1,T3,T4**
2. **yes, request is equal available (2,2,2,4)=(2,2,2,4)**
3. **yes, request is less than available (0,1,1,0)<(2,2,2,4)**
4. **yes, request is greater than available (2,2,1,2)>(2,2,2,4)**
5. A single-lane bridge connects the two Vermont villages of North Tunbridge and South Tunbridge. Farmers in the two villages use this bridge to deliver their produce to the neighboring town. The bridge can become deadlocked if a northbound and a southbound farmer get on the bridge at the same time. (Vermont farmers are stubborn and are unable to back up.).   
   Note that the bridge can’t holder more than one farmer at a time.
   1. Using semaphores and/or mutex locks, design an algorithm in pseudocode that prevents deadlock.   
      **Note:** Do not be concerned about starvation (the situation in which northbound farmers prevent southbound farmers from using the bridge, or vice versa).
   2. Design a monitor that prevents both deadlock and starvation.  
      Notes:
      * The bridge cannot hold more than one farmer. Thus, no farmer could enter the bridge while another is crossing it.
      * Northbound and southbound farmers should take turns in crossing the bridge. Two successive farmers from the same village could cross the bridge only if no farmers from the other village waiting for crossing the bridge

Answer :

1. semaphore cross = 1;

void enter\_bridge() {

P(cross);

}

void exit\_bridge() {

V(cross);

}

1. monitor bridge {

int num\_waiting\_north = 0;

int num\_waiting\_south = 0;

int on\_bridge = 0;

condition cross;

int prev = 0;

void enter\_bridge\_north() {

num\_waiting\_north++;

while (on\_bridge ||(prev == 0 && num\_waiting\_south >0)) ok\_to\_cross.wait();

on\_bridge=1;

num\_waiting\_north--;

prev = 0;

}

void exit\_bridge\_north() {

on\_bridge = 0;

cross.broadcast();

}

void enter\_bridge\_south() {

num\_waiting\_south++;

while (on\_bridge ||(prev == 1 && num\_waiting\_north > 0))

cross.wait();

on\_bridge=1;

num\_waiting\_south--;

prev = 1;

}

void exit\_bridge\_south() {

on\_bridge = 0;

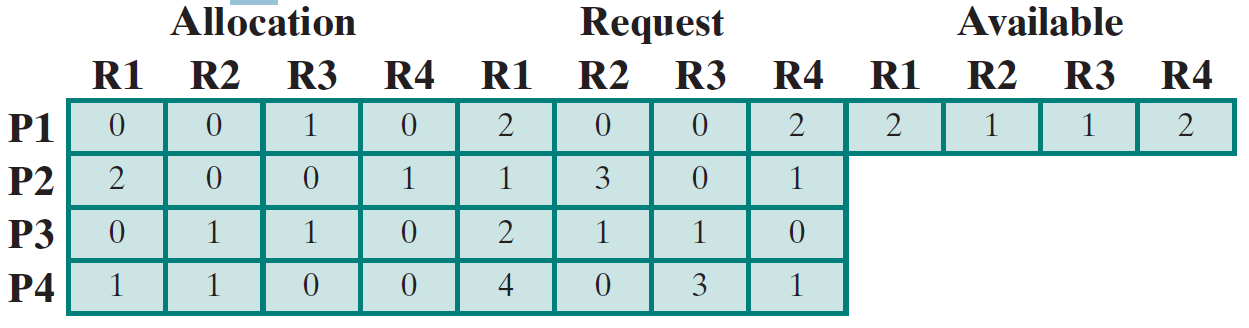
cross.broadcast();

}

}

1. At an instant, the resource allocation state in a system is as follows:
   * 4 processes P1–P4
   * 4 resource types: R1–R4
   * R1 (5 instances), R2 (3 instances), R3 (3 instances), R4 (3 instance)

Snapshot at time :



Run the deadlock detection algorithm and test whether the system is deadlocked or not. If it is, identify the processes that are deadlocked.

Answer : 1-P1 finishes and available (2,1,2,2)

2- p3 finishes and available (2,2,3,2)

The system will deaclock and processes are deadlockes : p2 and p4

1. In Section 8.5.4, we described a situation in which we prevent deadlock by ensuring that all locks are acquired in a certain order. However, we also point out that deadlock is possible in this situation if two threads simultaneously invoke the transaction() function. **Fix the transaction() function to prevent deadlocks.**

void transaction(Account from, Account to, double amount)

{

Semaphore lock1, lock2;

lock1 = getLock(from);

lock2 = getLock(to);

wait(lock1);

withdraw(from, amount);

signal(lock1);

wait(lock2);

deposit(to, amount);

signal(lock2);

}