CS & IT ENGINEERING

Operating System

Process Synchronization/Coordination

Classical IPC Problems

Lecture No. 7

By- Dr. Khaleel Khan Sir

- Producer-Consumer Problem
- Readers – Writer Problem
- Dining Philosopher Problem

Producer

Buffer

Consumer

Do you understand
This analogy of
Producer-consumer?

Producer needs to **WAIT**
If buffer is **FULL**
Consumer needs to **WAIT**
If buffer is **EMPTY**

```
#define N 100
int Buffer[N];
CSEM Empty = N;
```
< No. of Empty Slots in Buffer >
```
CSEM full = 0;
```
< No. of full Slots (data items) in Buffer >
```
BSEM mutex = 1;
```
< used by (P) & (C) to access Buffer as "CS" >

```
void Producer(void)
{
  int itemp, in=0;
  while(1)
  {
    a) itemp = Produce_item();
    b) DOWN(Empty);
    c) DOWN(mutex);
    d) Buffer[in] = itemp;
    e) in = (in+1) % N;
    f) UP(mutex);
    g) up(full);
  }
}
```

```
void Consumer(void)
{
  int itemc, out=0;
  while(1)
  {
    a) DOWN(full);
    b) DOWN(mutex);
    c) itemc = Buffer[out];
    d) out = (out+1) % N;
    e) up(mutex);
    f) up(Empty);
    g) Process_item(itemc);
  }
}
```

Q P40

Consider the following solution to the producer-consumer synchronization problem. The shared buffer size is N. Three semaphores empty, full and mutex are defined with respective initial values of 0, N and 1. Semaphore empty denotes the number of available slots in the buffer; for the consumer to read from. Semaphore full denotes the number of available slots in the buffer, for the producer to write to. The placeholder variables, denoted by P, Q, R and S, in the code below can be assigned either empty er full. The valid semaphore operations are: wait() and. sigma().

Producer:	Consumer:

do{
wait(P); P(full);
wait(mutex);
// Add item to buffer
Signal(mutex);
Signal(Q); } V(Empty);
} while (1);
	do{
	wait(R); P(Empty)
	wait(mutex);
	//Consume item from buffer
	signal(mutex);
	signal (S); V(full);
	} while (1);

Empty = 0; [No. of data items]
full = N;
mutex = 1;

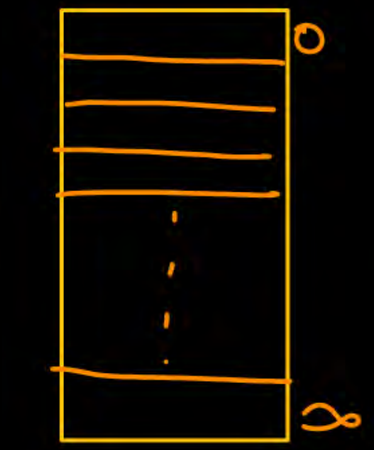
Which one of: the following assignments to P, Q, R and S will yield the correct solution?

(A) P: full, Q: full, R: empty, S: empty
(B) P: empty, Q: empty, R: full, S: full
(C) P: full, Q; empty, R: empty, S: full
(D) P: empty, Q: full, R: full, S: empty

Q) PYQ

Consider the procedure below for the Producer-Consumer problem which uses semaphores:

```
semaphore n = 0;
semaphore s = 1;
void producer()
{
    while(true)
    {
    produce ();
    semWait(s);
    addToBuffer();
    semSignal(s);
    semSignal(n);
    }
}
```




```
void consumer()
{
    while(true)
    {
    semWait(s);
    semWait(n);
    removeFromBuffer()
    semSignal(s);
    consume();
    }
}
```

Which one of the following is TRUE?

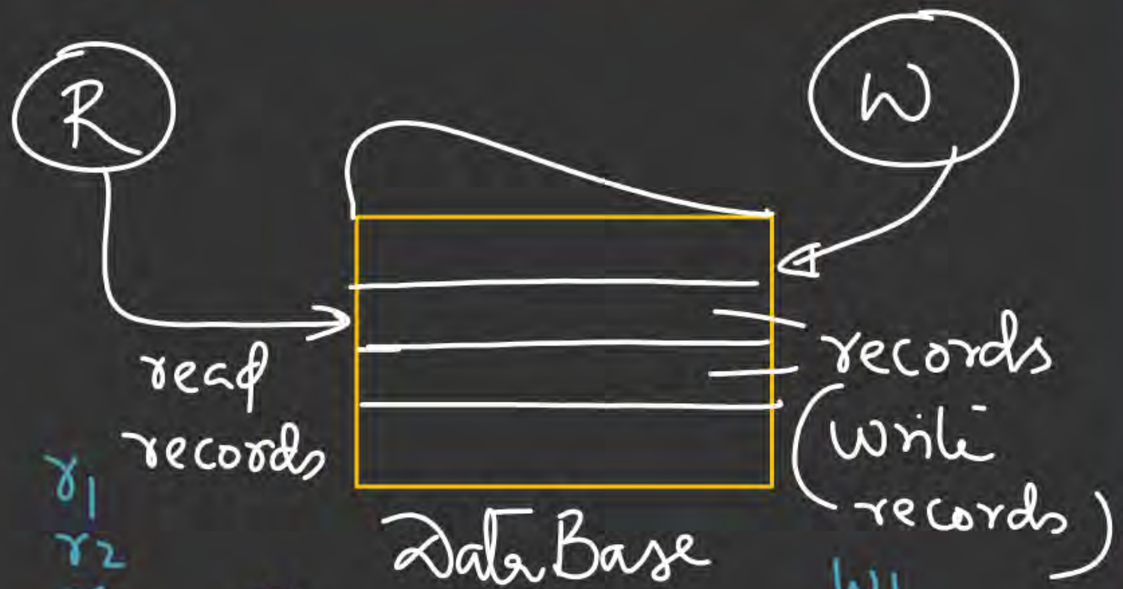
(A) The producer will be able to add an item to the buffer, but the consumer can never consume it.

(B) The consumer will remove no more than one item from the buffer.

(C) Deadlock occurs if the consumer succeeds in acquiring semaphore s when the buffer is empty.

(D) The starting value for the semaphore n must be 1 and not 0 for deadlock-free operation.

2) Reader-writer Problem :



(Reder & writer must access "DB" Mutually exclusion)

t_1: r_1 ✓ (or) w_1 ✓ : ✓

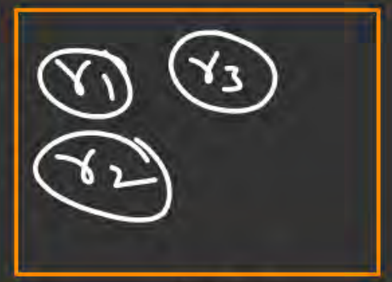
t_1: r_1 and w_1 : Anyone is allowed.

t_2: r_1 : ✓ : w_1 has to wait

t_3: r_2 : ✓

t_4: r_3 : ✓

t_5: w_2 : X



DB

< First R-W Problem >

Implementation of First R-W using Semaphores;

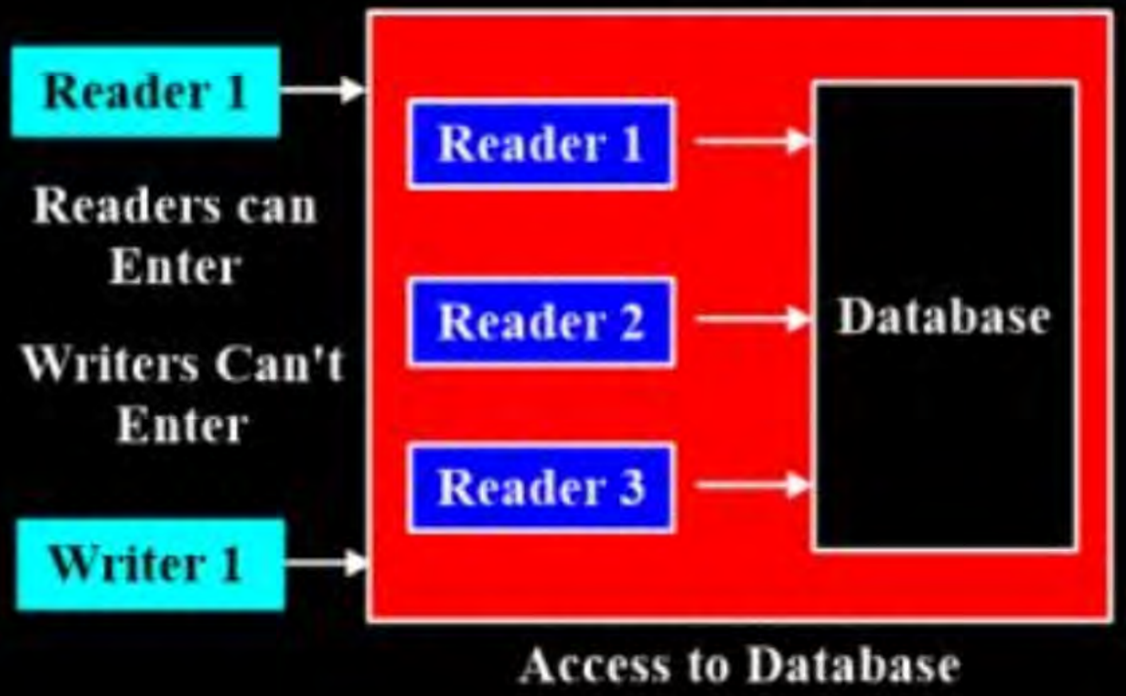
```
BSEM db = 1;
<used by (R) & (W) to access DB as CS>

int rc = 0;
<No. of (R's) Present in DB>

BSEM mutex = 1;
<used by (R's) to update "rc" in Mutually exclusive Control>
```

```
Void Writer(Void)
{
  while(1)
  {
    a) DOWN(db);
    b) <DB_WRITE>
    c) up(db);
  }
}
```

```
Void Reader(Void)
{
  while(1)
  {
    a) DOWN(mutex);
    b) rc = rc+1;
    c) if(rc==1) DOWN(db);
    d) up(mutex);
    e) <DB_READ>
    f) DOWN(mutex);
    g) rc = rc-1;
    h) if(rc==0) up(db);
    (i) up(mutex);
  }
}
```

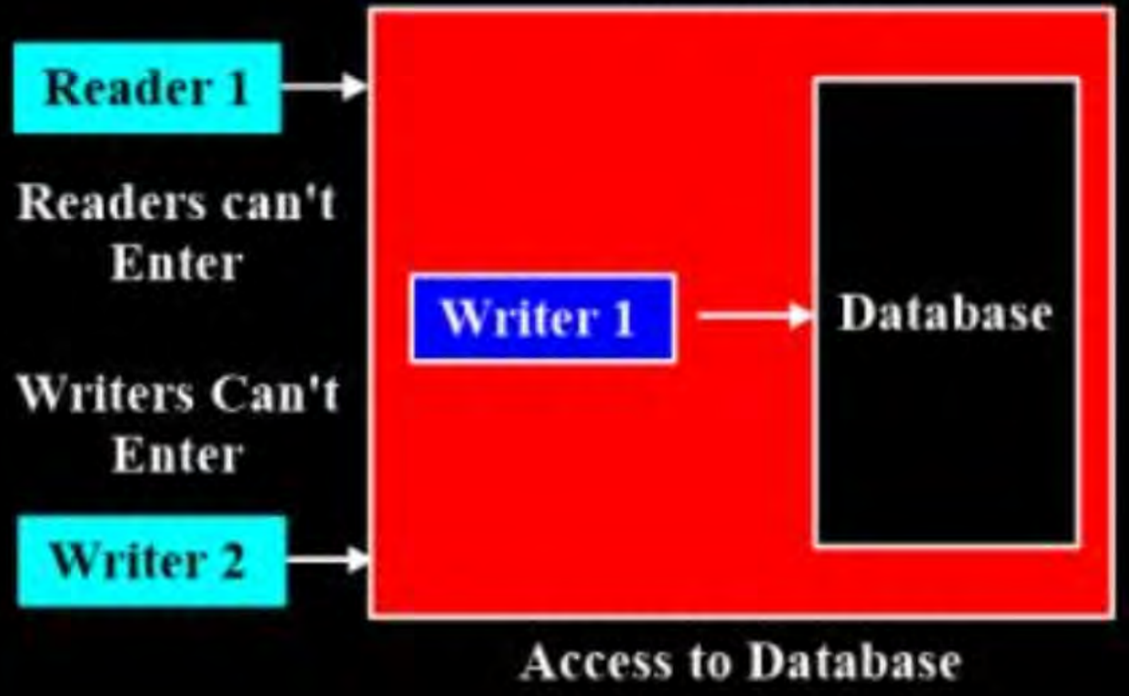
Readers Writers Operating System

When Readers are Accessing the Database



Readers Writers Operating System

When writer is writing in the Database



Q) Synchronization in the classical readers and writers problem can be achieved through use of semaphores. In the following incomplete code for readers-writers problem, two binary semaphores mutex and wrt are used to obtain synchronization

```
wait (wrt)
writing is performed
signal (wrt)
```

```
wait (mutex)
readcount = readcount + 1
if readcount = 1 then S1
S2
reading is performed
S3
readcount = readcount - 1
if readcount = 0 then S4
signal (mutex)
```

The values of S1, S2, S3, S4, (in that order) are

(A) signal (mutex), wait (wrt), signal (wrt), wait (mutex)

(B) signal (wrt), signal (mutex), wait (mutex), wait (wrt)

(C) wait (wrt), signal (mutex), wait (mutex), signal (wrt)

(D) signal (mutex), wait (mutex), signal (mutex), wait (mutex)

* 3) DINING PHILOSOPHERS Problem

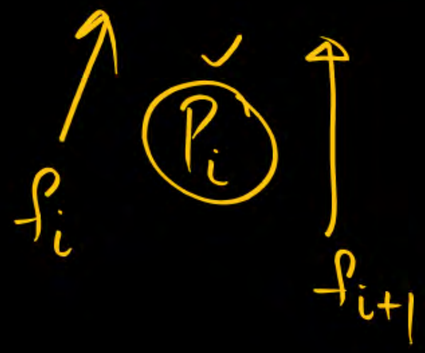
⟹ There are N-Philosophers

↳ Research Problem

↓

Solution

forks = N

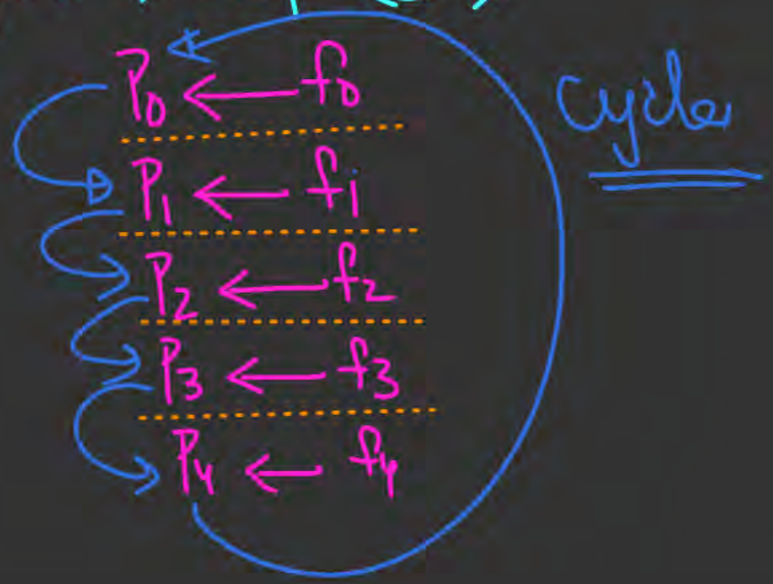


```
#define N 5
void Philosopher(int i)
{
    while (1)
    {
        a) Think(i);
        b) Take_fork(i);          L
        c) Take_fork((i+1)%N);    R
        d) <eat>
        e) put_fork(i);
        f) put_fork((i+1)%N);
    }
}
```

Is this Solution Correct ?

{Deadlock}

1) All (P_i's) Should become hungry
2) Each (P_i) must Preempt after step (b);



How to Prevent Deadlock?

Tanenbaum Text (Semaphore based Solution)

" Put the Semaphore control on accessing forks "

Non-Semaphore based:

1) Min. # of additional forks to avoid deadlock : 1 (Total ↓ N+1)

PYQ

Q) A solution to the Dining Philosophers Problem which avoids deadlock is:

(A) ensure that all philosophers pick up the left fork before the right fork ×

(B) ensure that all philosophers pick up the right fork before the left fork ×

(C) ensure that one particular philosopher picks up the left fork before the right fork, and that all other philosophers pick up the right fork before the left fork ✓

(D) None of the above

Q) Let m[0].....m[4] be mutexes (binary semaphore) and P0..P4 be processes. Suppose each process P[i] executes the following:

```
wait (m[i]) ; wait (m[i+1] mod 4)
CS
release (m[i]); release (m[i+1] mod 4)
```

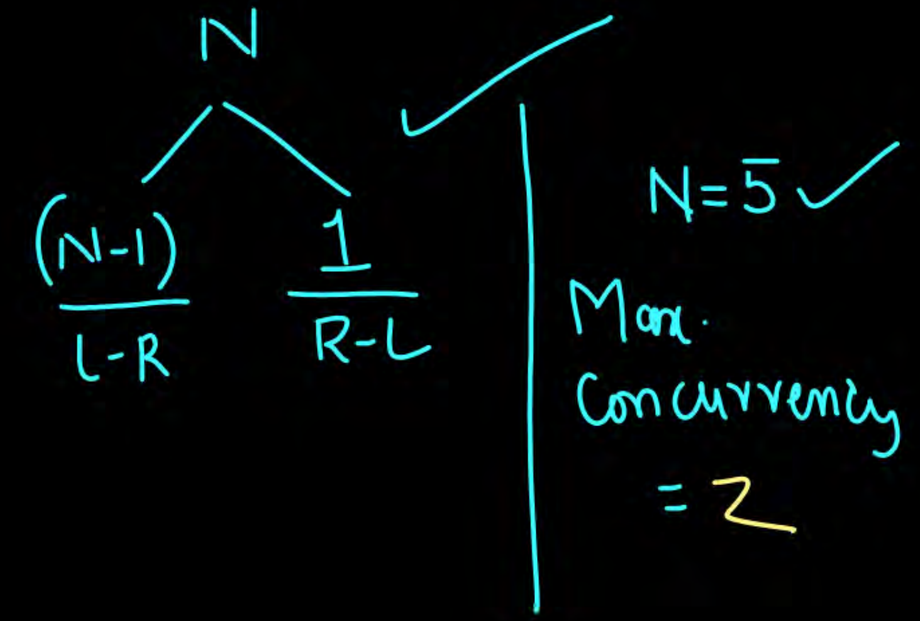
Which situation could be came

(a) Thrashing

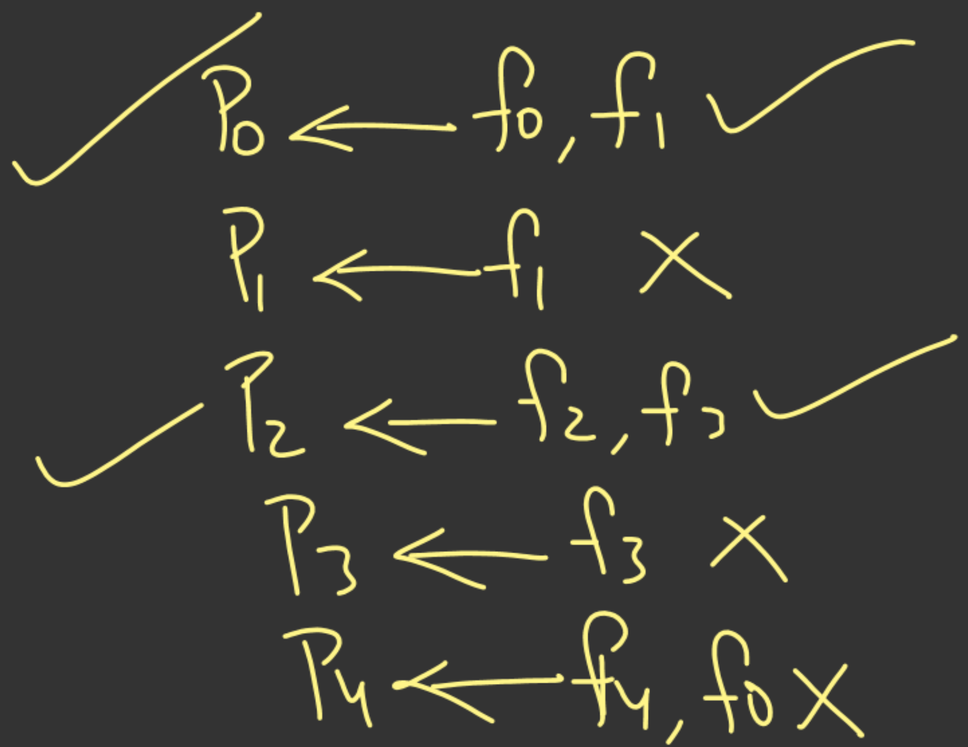
(b) Deadlock ✓

(c) Starvation but no deadlock

(d) None



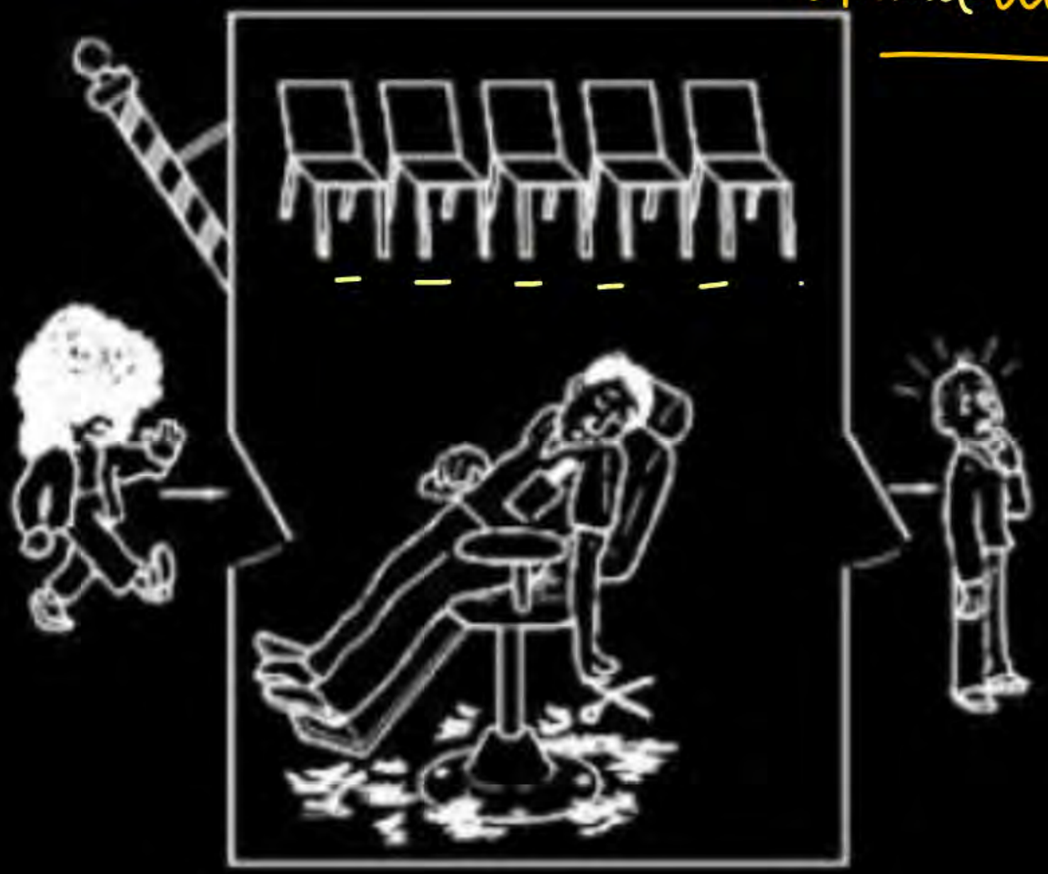
N = 5



Sleeping Barber Problem

(Tanenbaum) Barber - Shop Simulation

- There is one barber, and n chairs for waiting customers
- If there are no customers, then the barber sits in his chair and sleeps (as illustrated in the picture)
- When a new customer arrives and the barber is sleeping, then he will wakeup the barber
- When a new customer arrives, and the barber is busy, then he will sit on the chairs if there is any available, otherwise (when all the chairs are full) he will leave.



Barber Shop

Waiting Room

Barber has a chair to cut hair of customer. When barber is done with customer then he will cut hair of next customer from waiting room.


Barber Shop
Waiting Room
z z z

First Reader-Writer using Semaphore with Busy Waiting

```
int R = 0, W = 0;
Bsem mutex = 1;
Void Reader (Void)
{
  L1: P(mutex);
  if (W == = 1)
  {
     V(mutex);
     goto L1;
  }
  else
  {
     R = R+1;
     V(mutex);
  }
<DB_READ>
P(mutex);
   R = R –1;
V(mutex);
}
```

```
Void Writer ( Void)
{
  L2: P(mutex);
  if(_____) ⇒ [R ≥ 0 (or) W = 1]
  {
       V(mutex);
       goto L2;
  }
  else
  {
       W=1;
       V(mutex);
  }
  <DB_WRITE>
  P(mutex);
       W=0;
  V(mutex);
}
```


THANK YOU
GW
SOLDIERS !