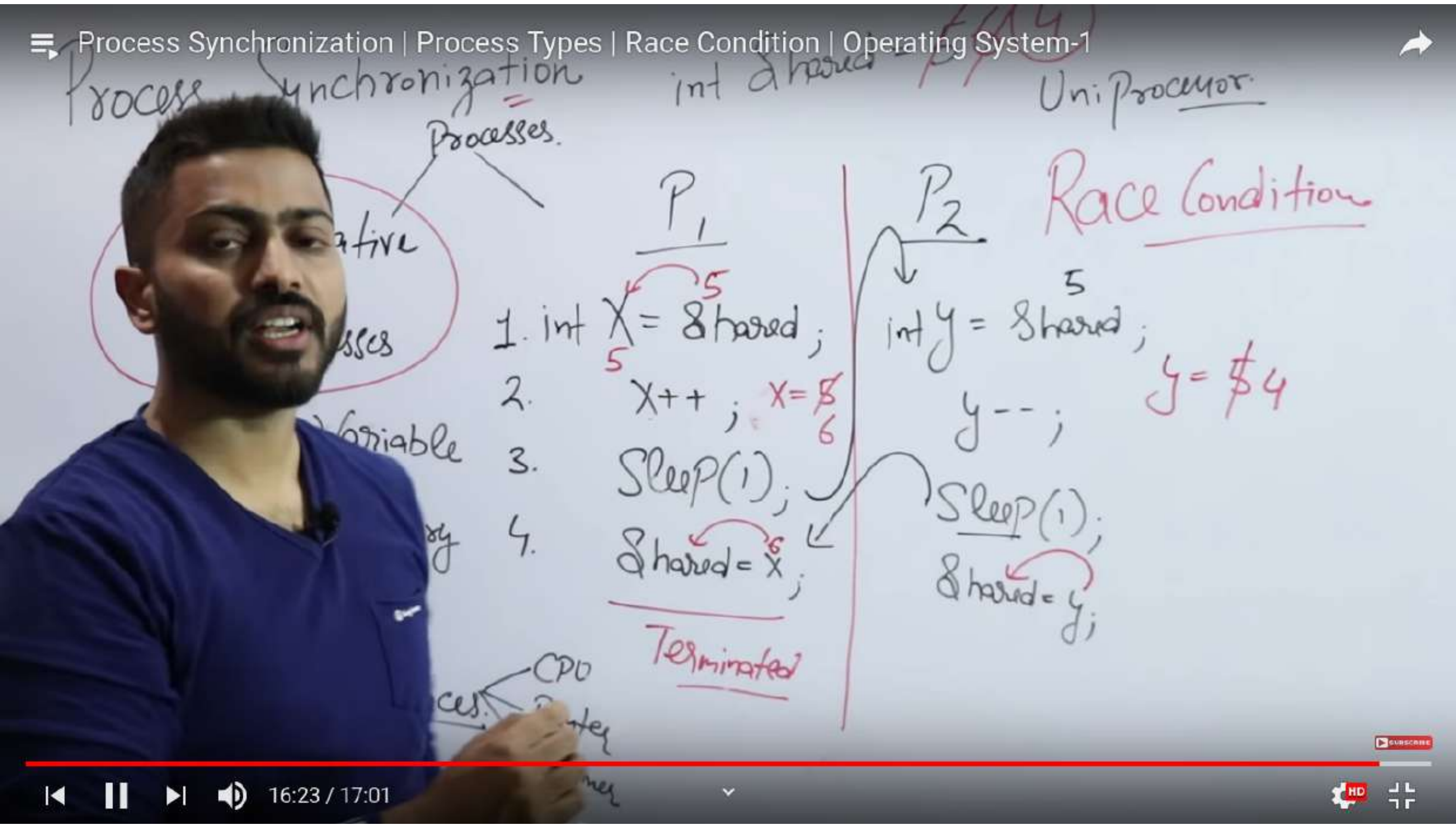
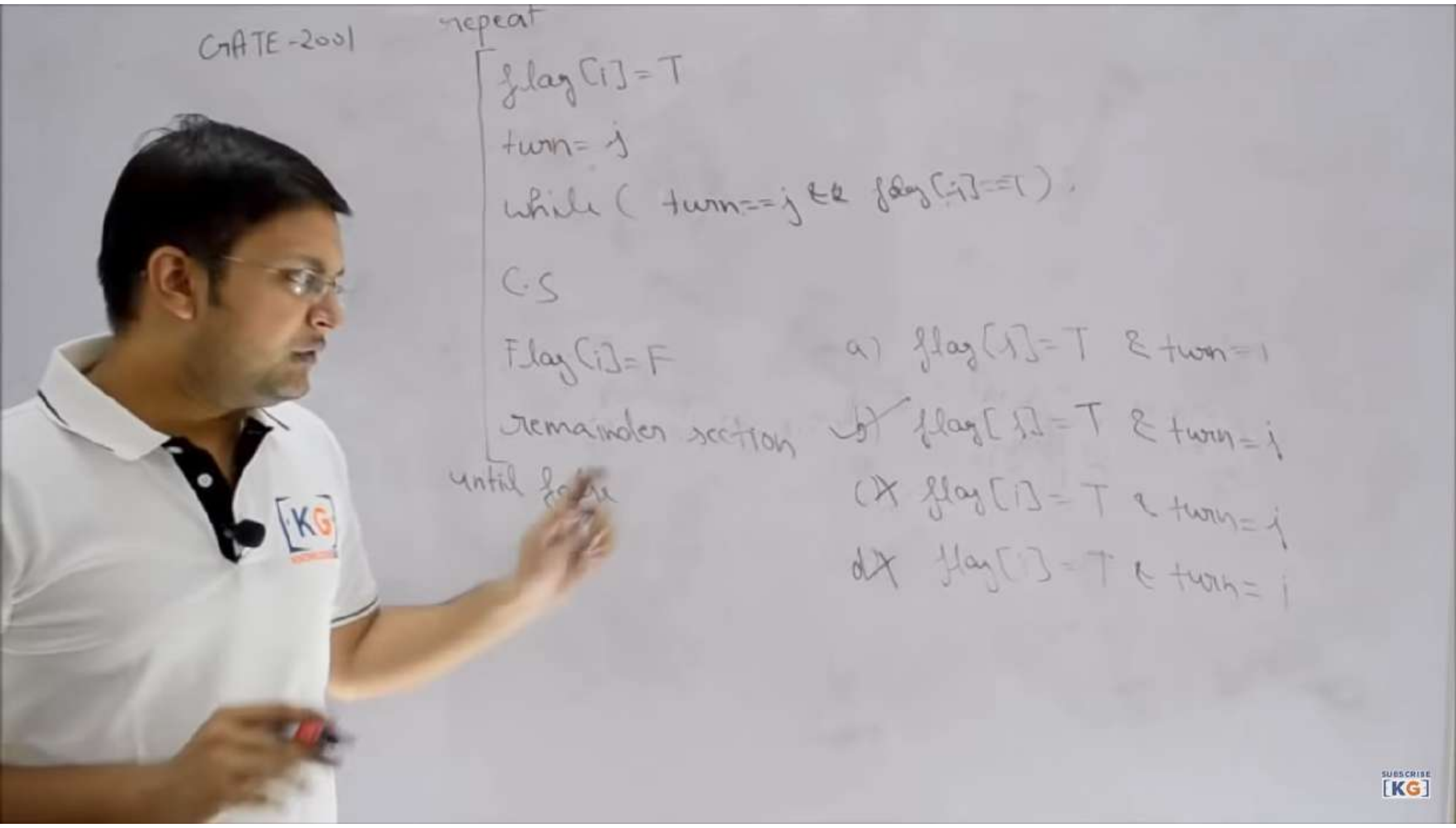
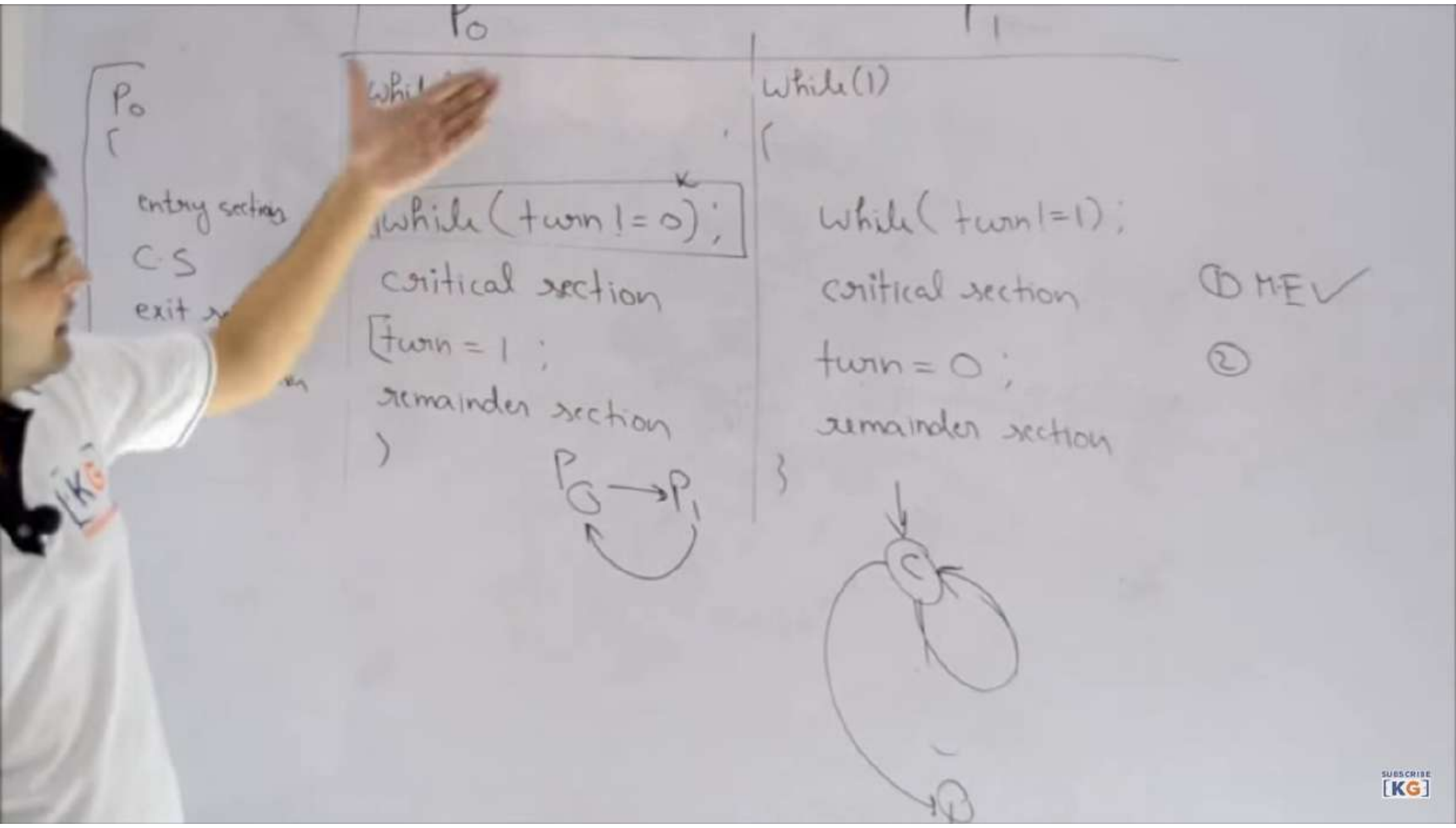
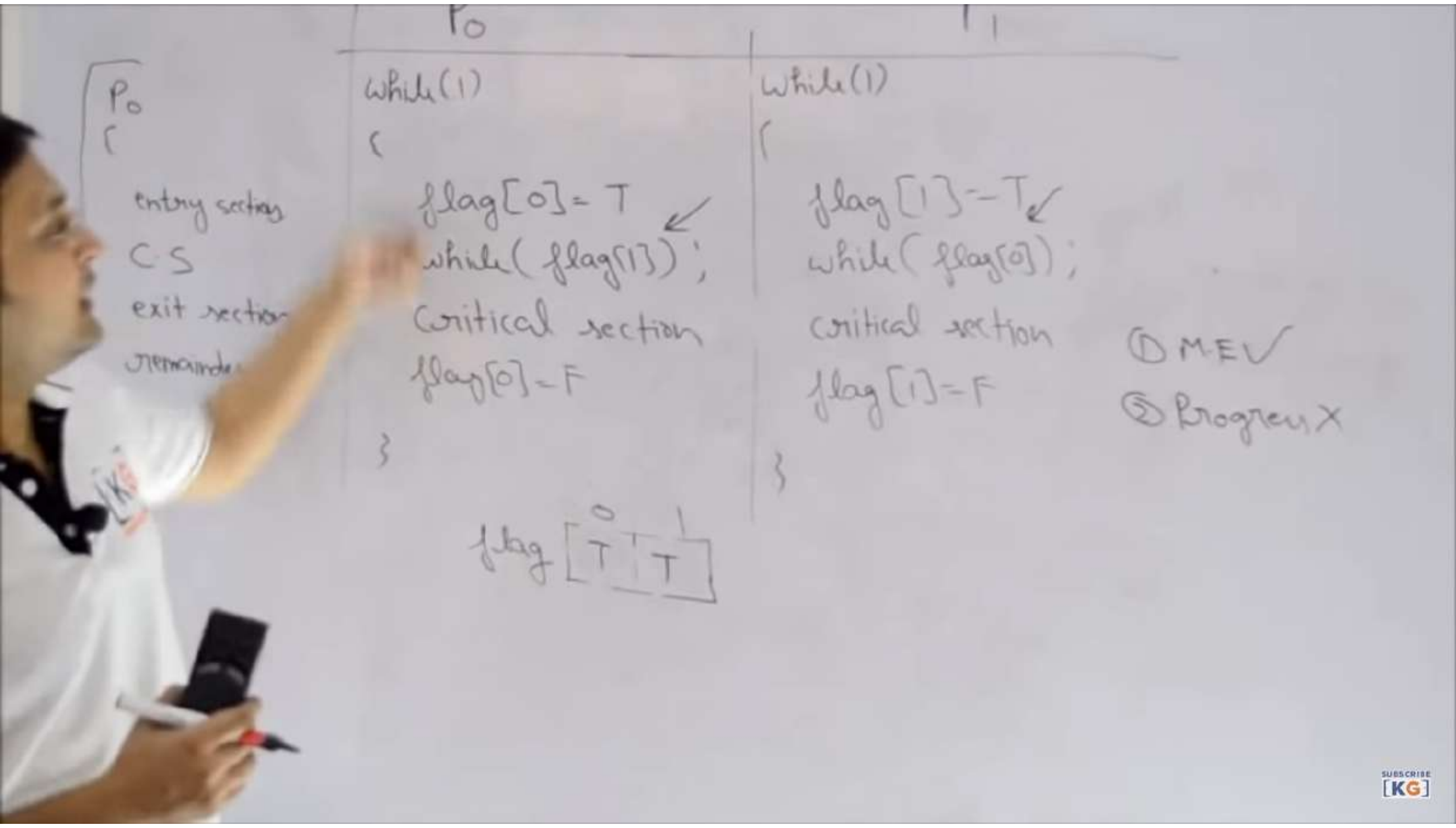


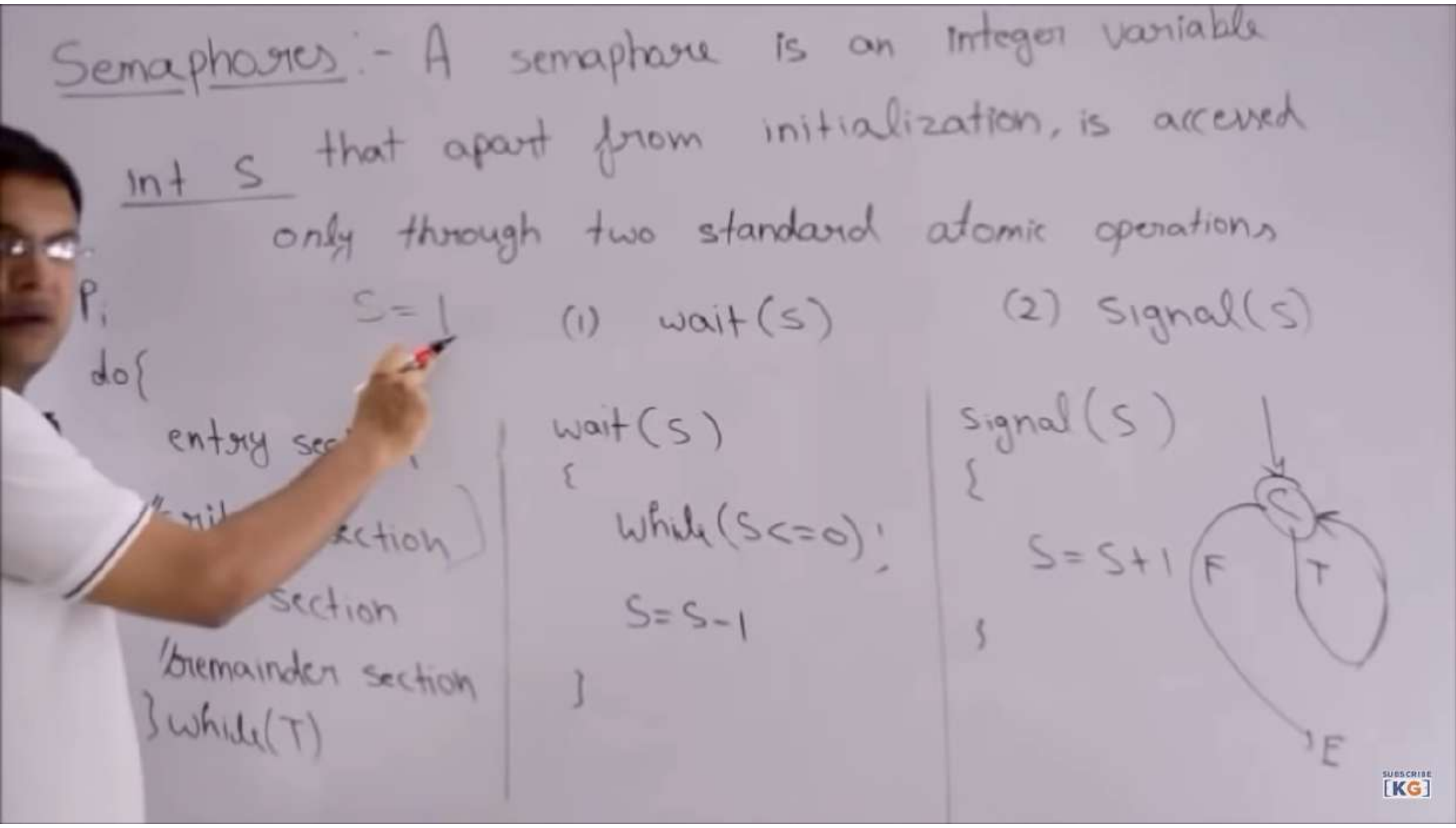
Process Synchronization | Process Types | Race Condition | Operating System-1
Process Synchronization
Processes
int Shared
Uni Processor
Race Condition
P1
P2
1. int X = Shared;
2. X++;
3. Sleep(1);
4. Shared = X;
Terminated
int y = Shared;
y--;
Sleep(1);
Shared = y;
Variable
CPU
16:23 / 17:01

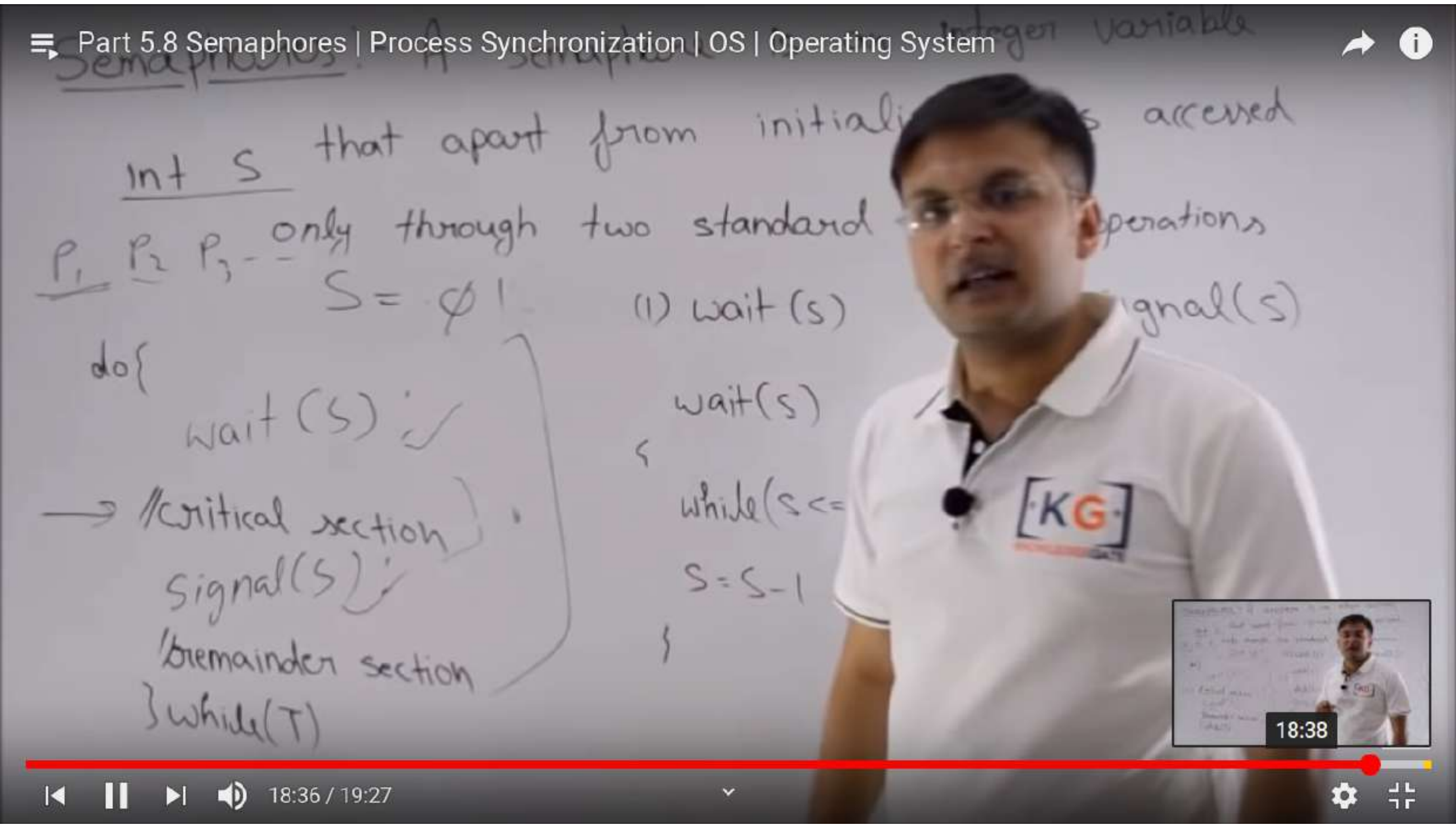

GATE-2001
repeat
flag[i] = T
turn = j
while (turn==j && flag[j]==T) ;
C.S
Flag[i] = F
remainder section
until false
a) flag[j] = T & turn = i
b) flag[j] = T & turn = j
c) flag[i] = T & turn = j
d) flag[i] = T & turn = i
SUBSCRIBE
KG


P0
P1
P0
{
entry section
C.S
exit
while(turn != 0);
critical section
turn = 1 ;
remainder section
}
P0 → P1
while(1)
{
while(turn!=1);
critical section
turn = 0 ;
remainder section
}
① M.E ✓
②
SUBSCRIBE
KG

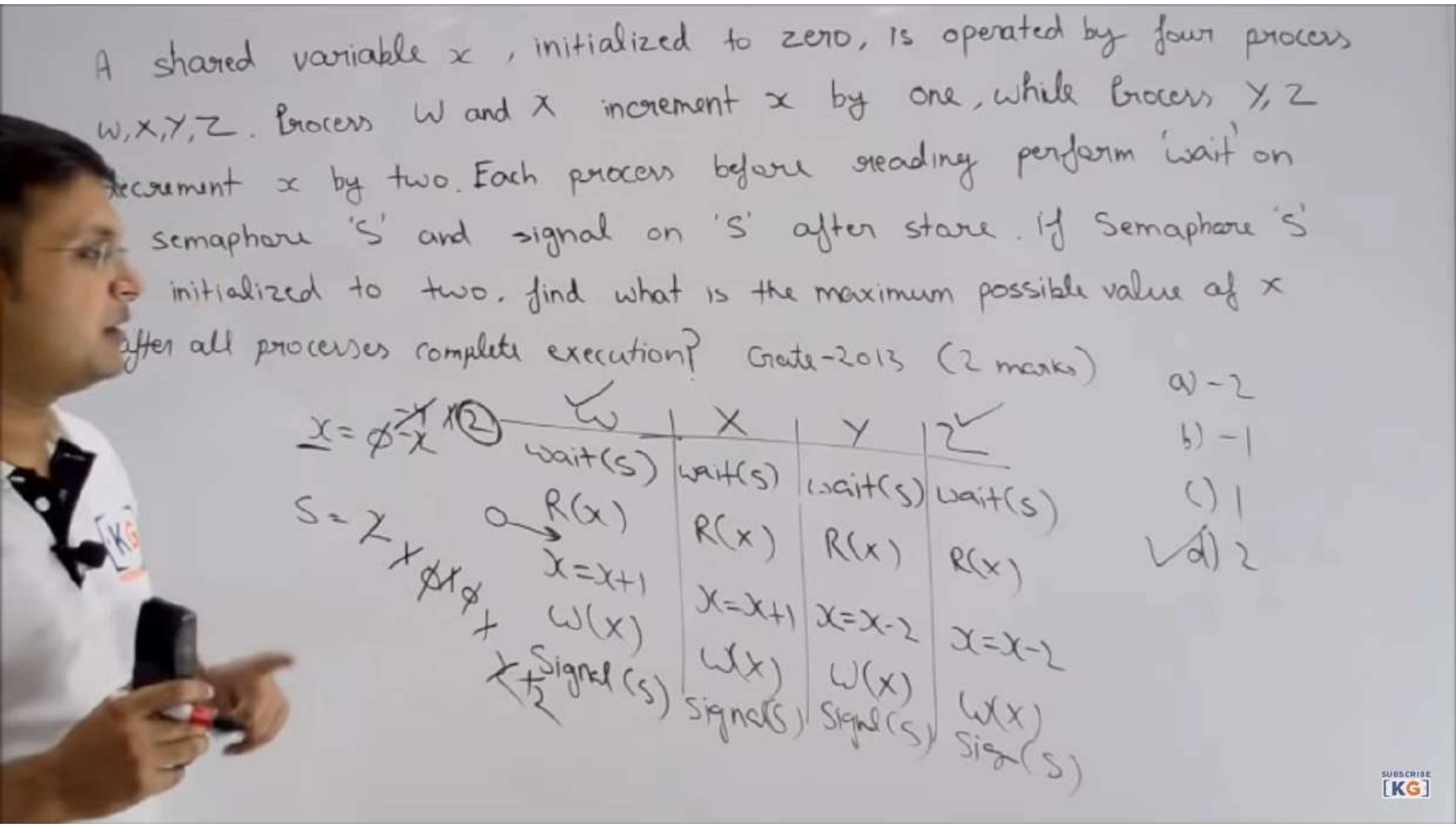


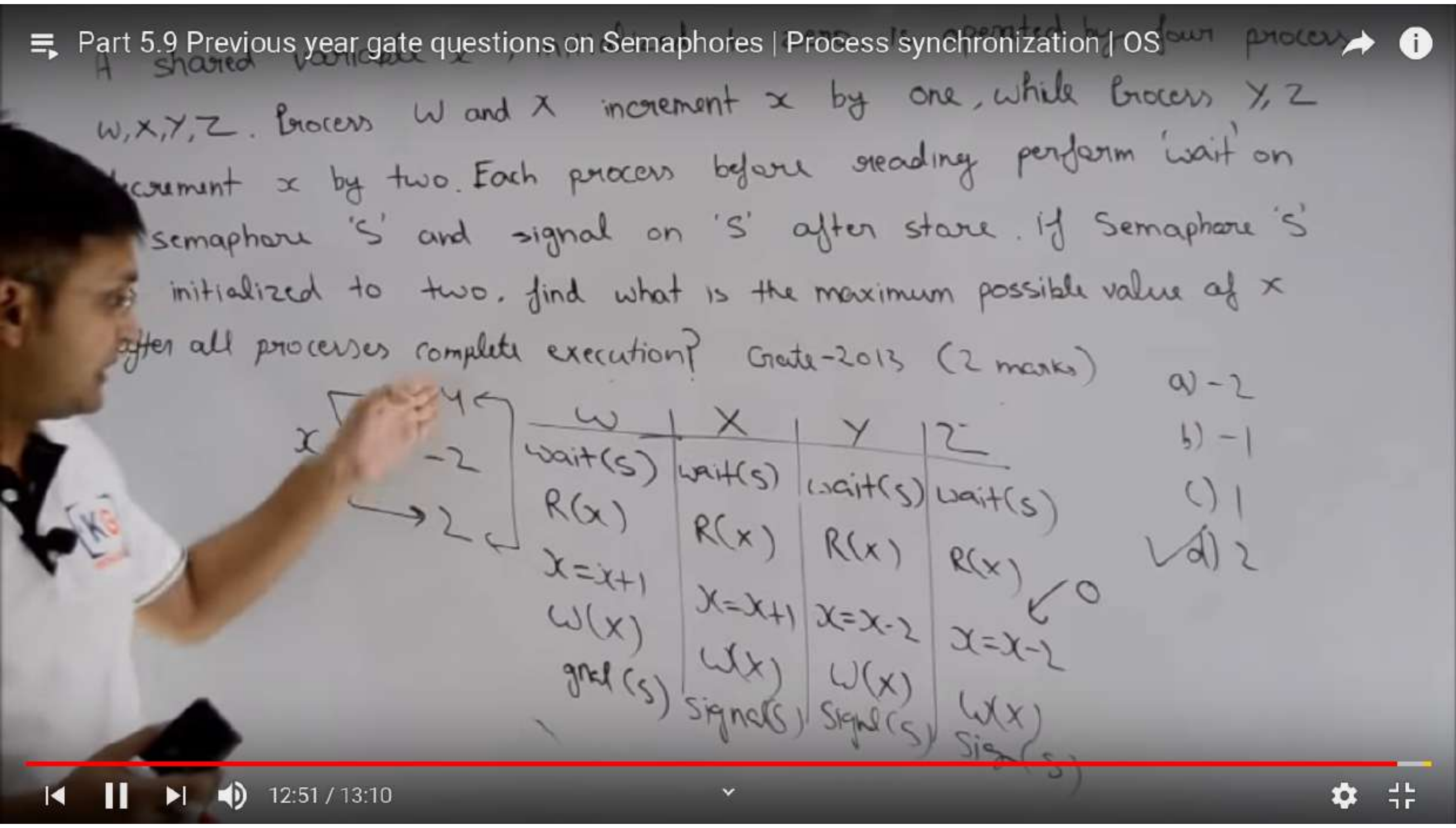
P0
P1
P0
{
entry section
C.S
exit section
remainder
while(1)
{
flag[0] = T
while(flag[1]);
critical section
flag[0] = F
}
while(1)
{
flag[1] = T
while(flag[0]);
critical section
flag[1] = F
}
flag [T | T]
0
1
① M.E ✓
② Progress ✗
SUBSCRIBE
KG

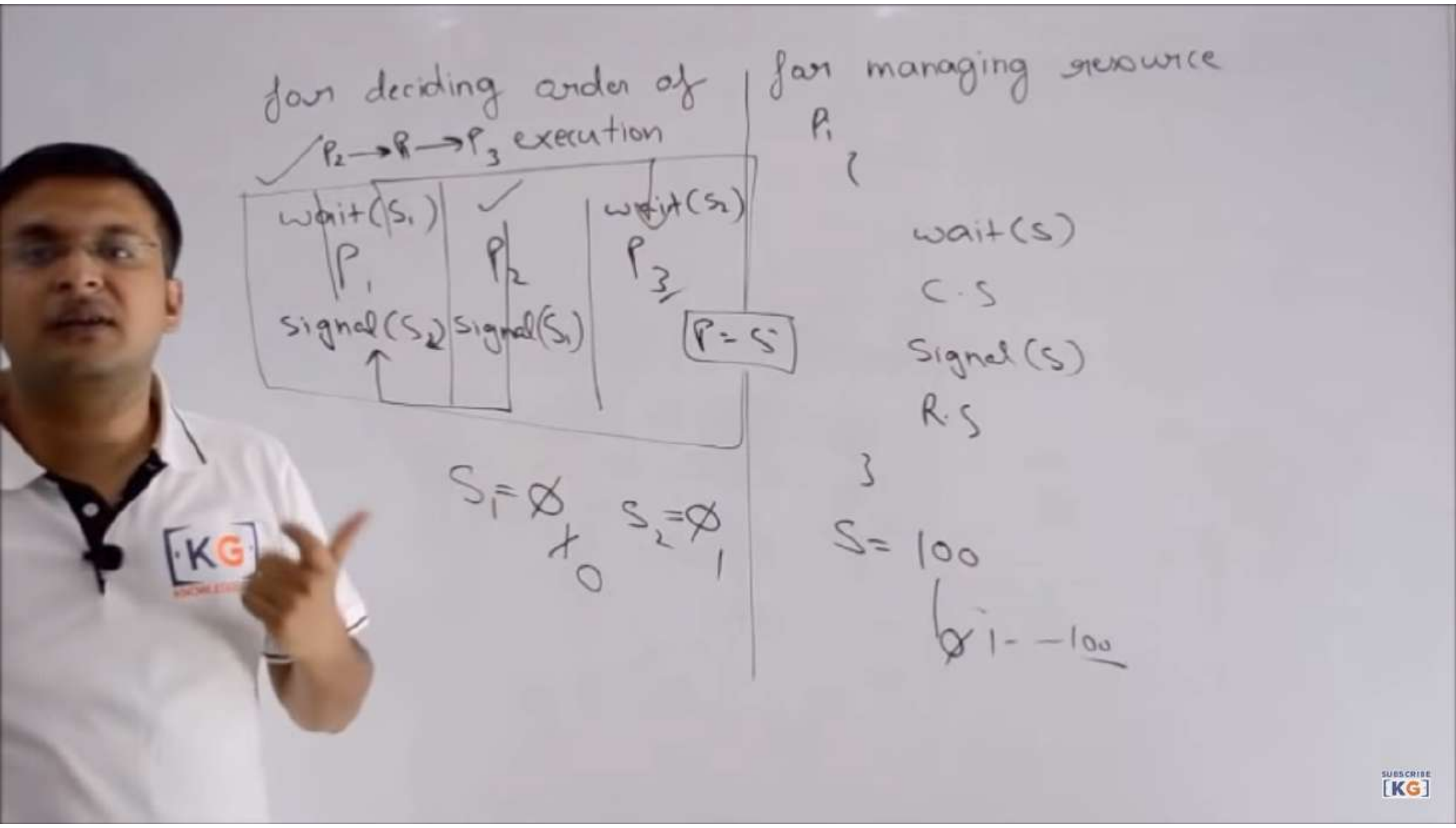

Semaphores :- A semaphore is an integer variable
int S that apart from initialization, is accessed
only through two standard atomic operations
Pi
S=1
(1) wait(S)
(2) signal(S)
do{
entry section
section
remainder section
}while(T)
wait(S)
{
while(S<=0);
S=S-1
}
signal(S)
{
S=S+1
}
S
F
T
E



Part 5.8 Semaphores | Process Synchronization | OS | Operating System
int S that apart from
accessed
P_1 P_2 P_3 - - only through two standard
operations
S = Ø !
(1) wait (S)
do{
wait (S);
//critical section
signal(S);
//remainder section
}while(T)
wait(S)
{
S=S-1
}
18:38
18:36 / 19:27


A shared variable x, initialized to zero, is operated by four process W, X, Y, Z. Process W and X increment x by one, while process Y, Z decrement x by two. Each process before reading perform 'wait' on semaphore 'S' and signal on 'S' after store. If Semaphore 'S' initialized to two, find what is the maximum possible value of x after all processes complete execution? Gate-2013 (2 marks)
a) -2
b) -1
c) 1
d) 2
x = 0 -2 -1 2
S = 2 1 0 1 0 1 2
W	X	Y	Z
wait(S)	wait(S)	wait(S)	wait(S)
R(x)	R(x)	R(x)	R(x)
x=x+1	x=x+1	x=x-2	x=x-2
W(x)	W(x)	W(x)	W(x)
Signal(S)	Signal(S)	Signal(S)	Signal(S)


Part 5.9 Previous year gate questions on Semaphores | Process synchronization | OS
A shared variable x, ... four process
W, X, Y, Z. Process W and X increment x by one, while Process Y, Z
decrement x by two. Each process before reading perform 'wait' on
semaphore 'S' and signal on 'S' after store. If Semaphore 'S'
is initialized to two. find what is the maximum possible value of x
after all processes complete execution? Gate-2013 (2 marks)
a) -2
b) -1
c) 1
d) 2
W
X
Y
Z
wait(S)
R(x)
x=x+1
W(x)
signal(S)
x=x-2
12:51 / 13:10


for deciding order of
P2 → P1 → P3 execution
wait(S1)
P1
signal(S2)
P2
signal(S1)
wait(S2)
P3
P = S
S1 = Ø
S2 = Ø
for managing resource
P1
{
wait(S)
C.S
signal(S)
R.S
}
S = 100
1 - - 100

Part 5.11 Previous year practise questions on Semaphores | Process synchronization | OS |

Suppose we want to synchronize ... using binary semaphores S, T.

o/p string should be like '001100110011'

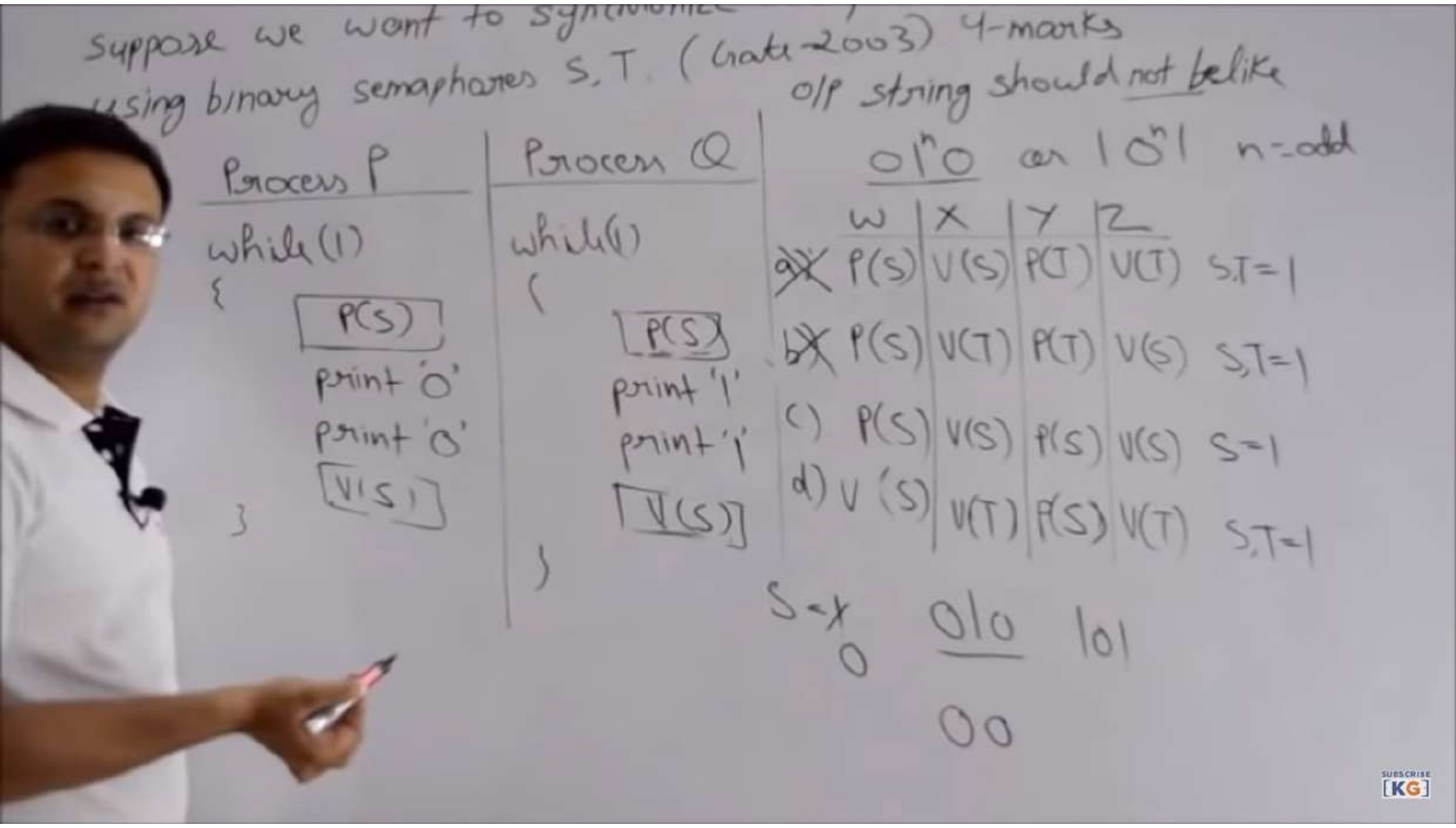
Process P	Process Q
while(1)	while(1)
{	{
$\boxed{P(S)}$	$\boxed{P(T)}$
print '0'	print '1'
print '0'	print '1'
$\boxed{V(T)}$	$\boxed{V(S)}$
}	}

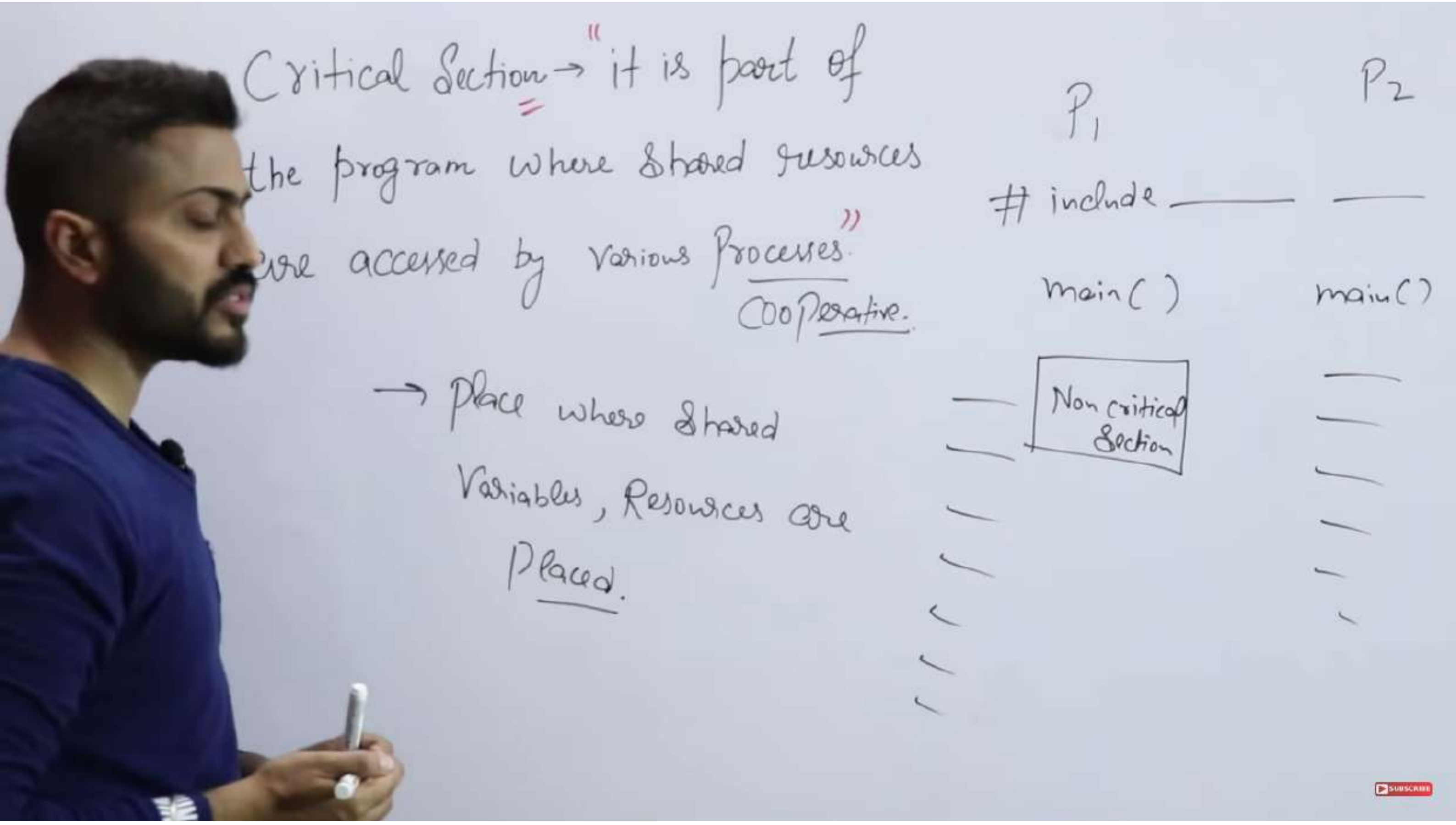
	W	X	Y	Z	
~~a)~~	P(S)	V(S)	P(T)	V(T)	S,T=1
b)	P(S)	V(T)	P(T)	V(S)	S=1, T=0
c)	P(S)	V(T)	P(T)	V(S)	S,T=1
d)	P(S)	V(S)	P(T)	V(T)	S=1, T=0

$S = \cancel{1}\,\cancel{0}\,1$; $T = \cancel{0}\,\cancel{1}\,0$

001100

7:07 / 14:26


suppose we want to synchronize
(Gate-2003) 4-marks
using binary semaphores S, T.
o/p string should not be like
01^n0 or 10^n1 n=odd
Process P
while(1)
{
P(S)
print '0'
print '0'
V(S)
}
Process Q
while(1)
{
P(S)
print '1'
print '1'
V(S)
}
W | X | Y | Z
a) P(S) | V(S) | P(T) | V(T) S,T=1
b) P(S) | V(T) | P(T) | V(S) S,T=1
c) P(S) | V(S) | P(S) | V(S) S=1
d) V(S) | V(T) | P(S) | V(T) S,T=1
S=1
0
010
101
00


Critical Section → "it is part of
the program where shared resources
are accessed by various Processes"
cooperative.
→ Place where shared
Variables, Resources are
Placed.
P1
P2
# include
main( )
main( )
Non critical Section

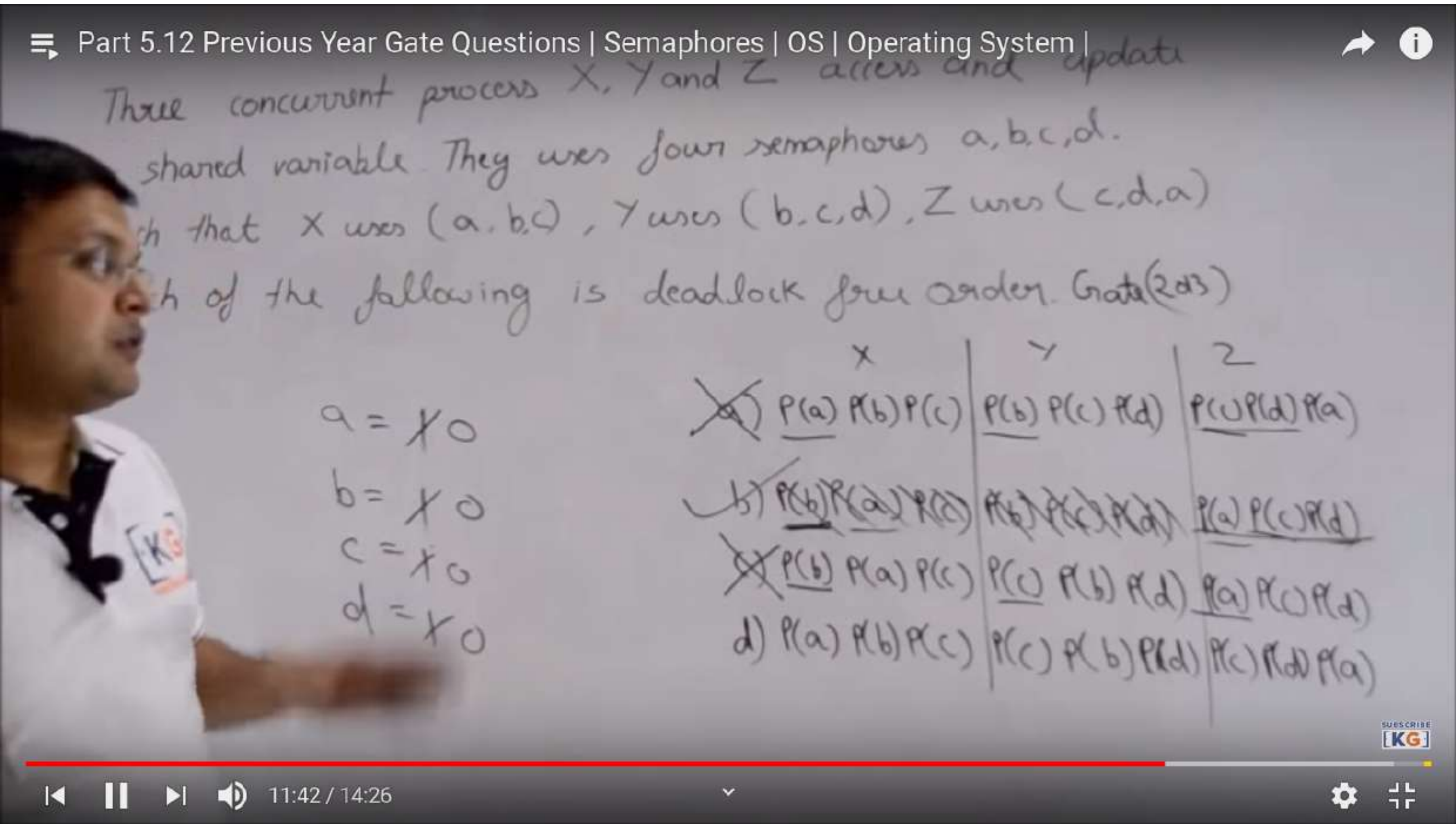
Three concurrent process X, Y and Z access and update shared variable. They uses four semaphores a, b, c, d. ich that X uses (a, b, c), Y uses (b, c, d), Z uses (c, d, a) hich of the following is deadlock free order. Gate(2013)

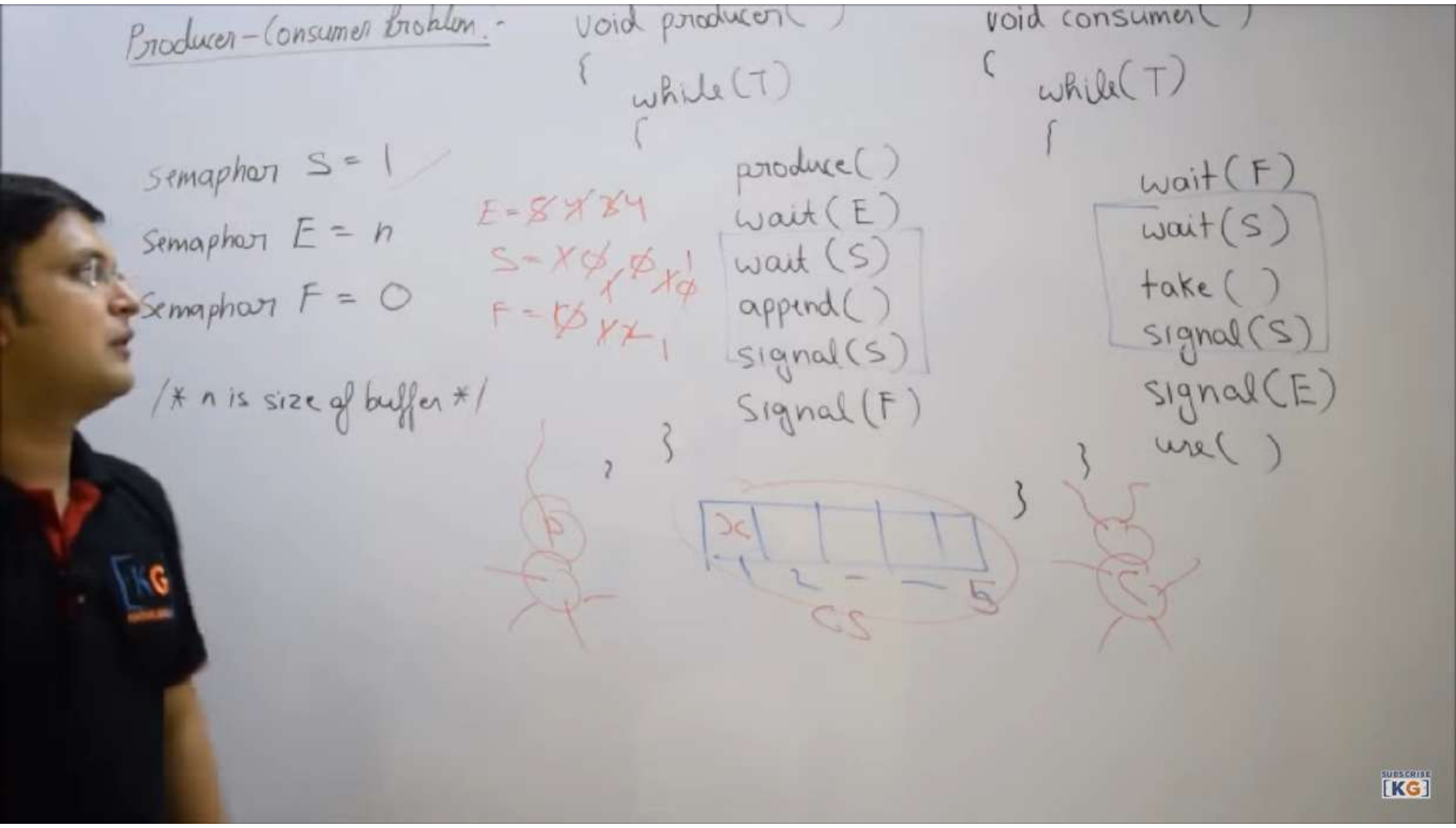
$X = X_0$
$Y = X_0$

P	Q
P(~~XX~~	P(Y) X
P(Y)	P(X)
C.S	C.S
V(X)	V(X)
V(Y)	V(Y)

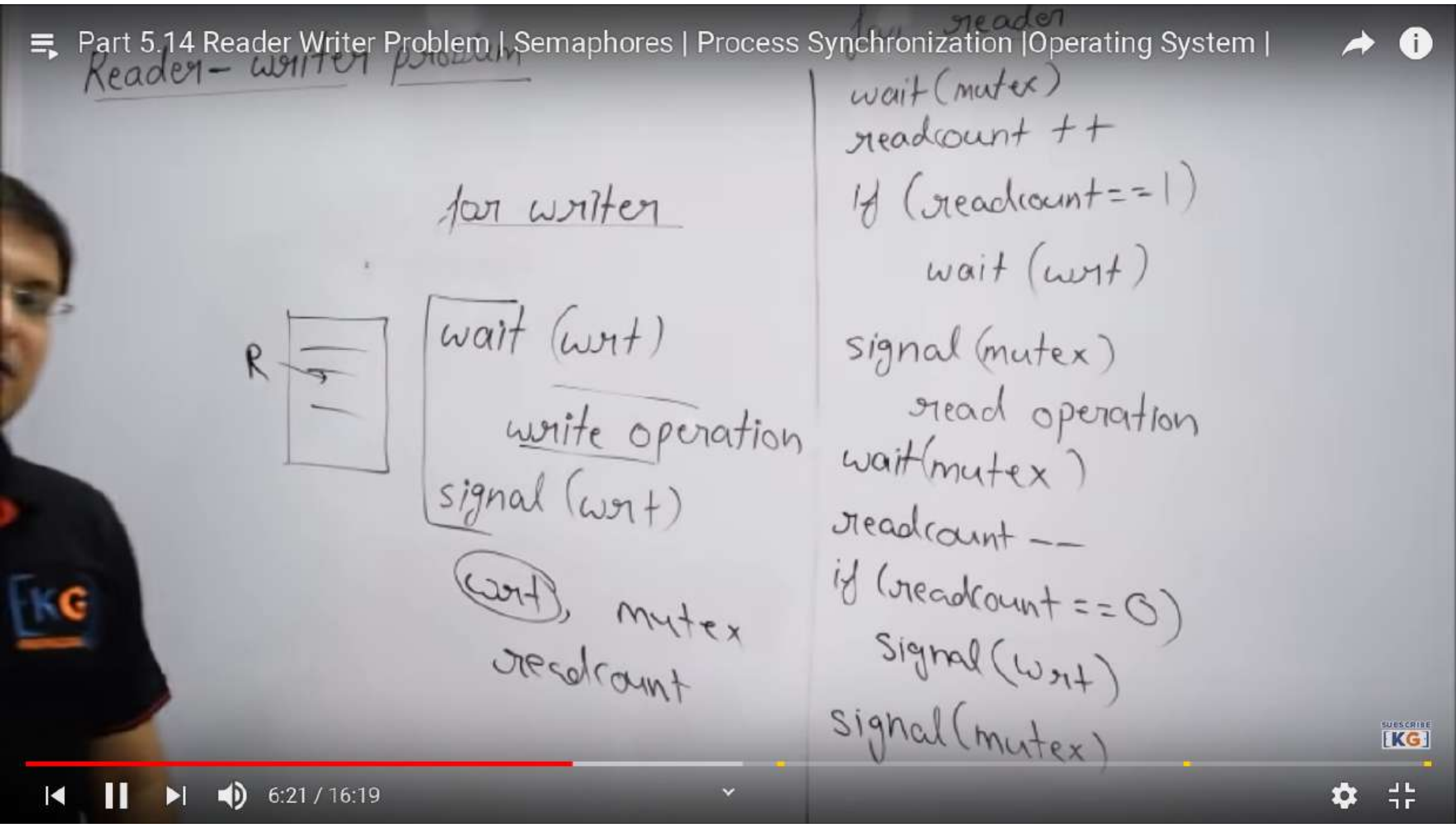
	X	Y	Z
a)	P(a) P(b) P(c)	P(b) P(c) P(d)	P(c) P(d) P(a)
b)	P(b) P(a) P(c)	P(b) P(c) P(d)	P(a) P(c) P(d)
c)	P(b) P(a) P(c)	P(c) P(b) P(d)	P(a) P(c) P(d)
d)	P(a) P(b) P(c)	P(c) P(b) P(d)	P(c) P(d) P(a)

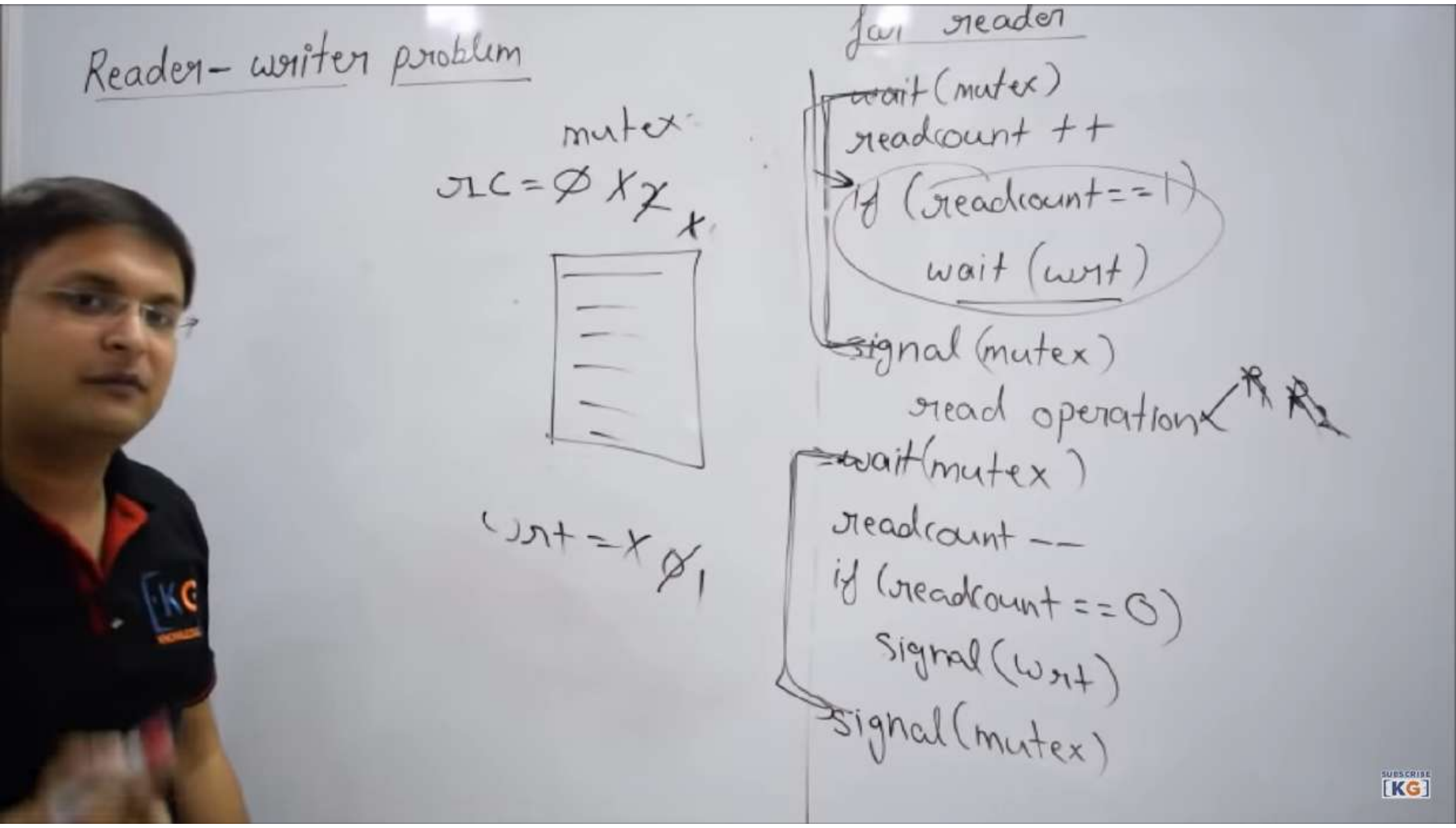
SUBSCRIBE KG


Part 5.12 Previous Year Gate Questions | Semaphores | OS | Operating System |
Three concurrent process X, Y and Z access and update
shared variable They uses four semaphores a, b, c, d.
ch that X uses (a, b, c), Y uses (b, c, d), Z uses (c, d, a)
h of the following is deadlock free order. Gate (2013)
a = 1 0
b = 1 0
c = 1 0
d = 1 0
X
Y
Z
a) P(a) P(b) P(c) | P(b) P(c) P(d) | P(c) P(d) P(a)
b) P(b) P(a) P(c) | P(b) P(c) P(d) | P(a) P(c) P(d)
c) P(b) P(a) P(c) | P(c) P(b) P(d) | P(a) P(c) P(d)
d) P(a) P(b) P(c) | P(c) P(b) P(d) | P(c) P(d) P(a)
11:42 / 14:26



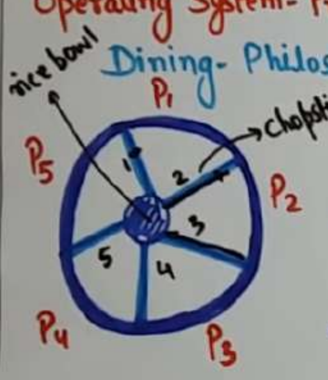
Producer-Consumer Problem:-
semaphor S = 1
Semaphor E = n
Semaphor F = 0
/* n is size of buffer */
void producer( )
{
while(T)
{
produce( )
wait(E)
wait (S)
append( )
signal(S)
Signal(F)
}
}
void consumer( )
{
while(T)
{
wait(F)
wait(S)
take( )
signal(S)
signal(E)
use( )
}
}
P
C
x
1
2
5
CS


Part 5.14 Reader Writer Problem | Semaphores | Process Synchronization |Operating System |
Reader- writer problem
for writer
R
wait (wrt)
write operation
signal (wrt)
wrt, mutex
readcount
for reader
wait(mutex)
readcount ++
if (readcount==1)
wait (wrt)
signal (mutex)
read operation
wait(mutex)
readcount --
if (readcount==0)
Signal (wrt)
Signal (mutex)
6:21 / 16:19


Reader- writer problem
mutex
rc = 0
wrt =
for reader
wait(mutex)
readcount ++
if (readcount==1)
wait (wrt)
signal (mutex)
read operation
wait(mutex)
readcount --
if (readcount == 0)
signal(wrt)
signal(mutex)
KG

Operating System- Process Synchronization

Dining- Philosophers Problem



Five Philosophers are sitting around a circular table.

- Dining table has five chopsticks and bowl of rice in the middle.
- Philosopher either can eat or think.
- When a philosopher wants to eat, he uses two chopsticks.
- When Philosopher wants to think, he keeps down both chopsticks.
- Problem is "No Philosopher will Starve".

[Each philosopher can forever continue to think and eat alternatively. It is assumed that no philosopher can know when others wants to eat or think.]

Deadlock May occur.

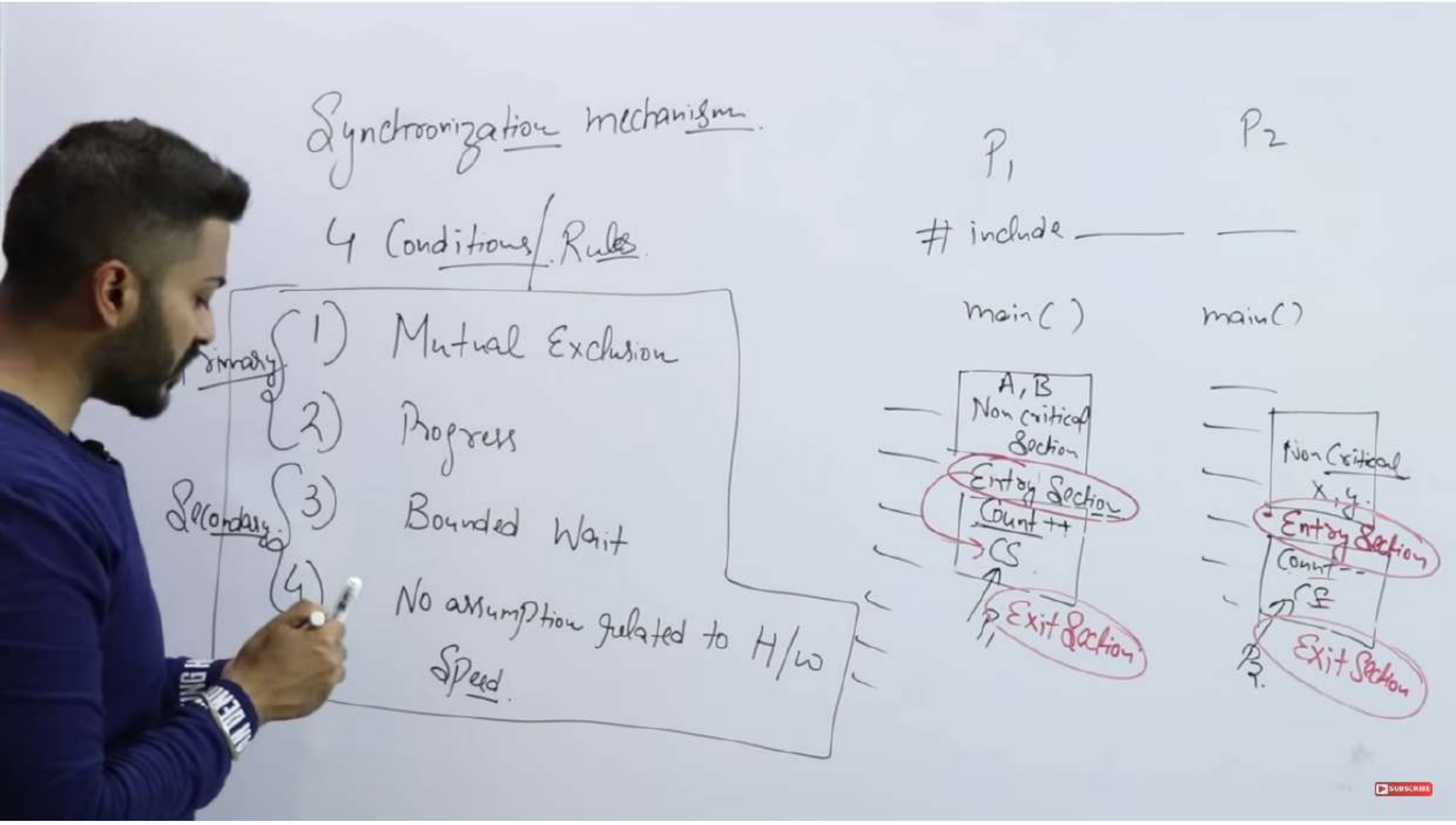
Chopstick [i];
↳ 5

Sol^n:- (5+1)%5 : 1

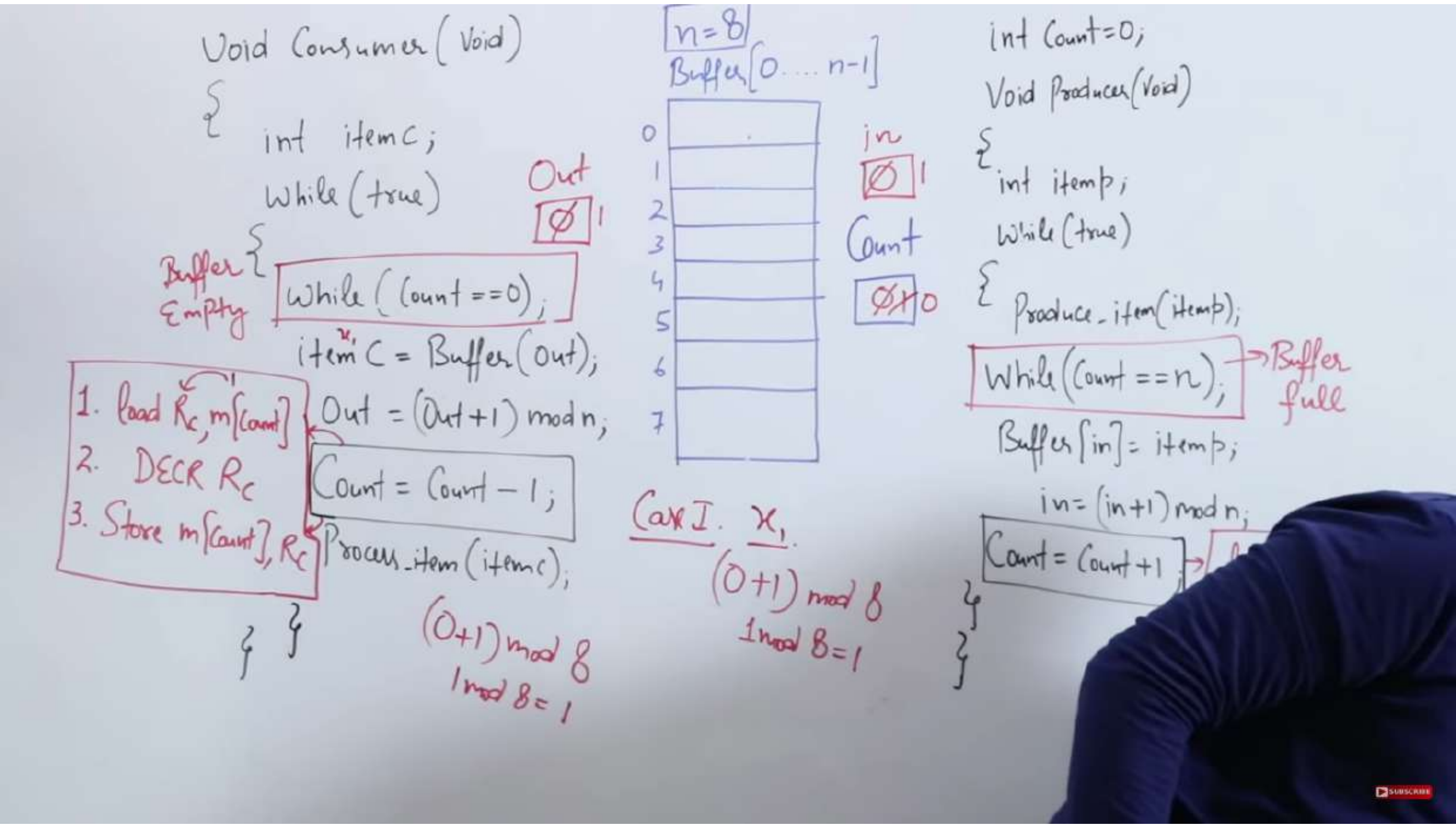
↳ i) Philosopher must be allowed to pick up chopsticks if both left and right are available.

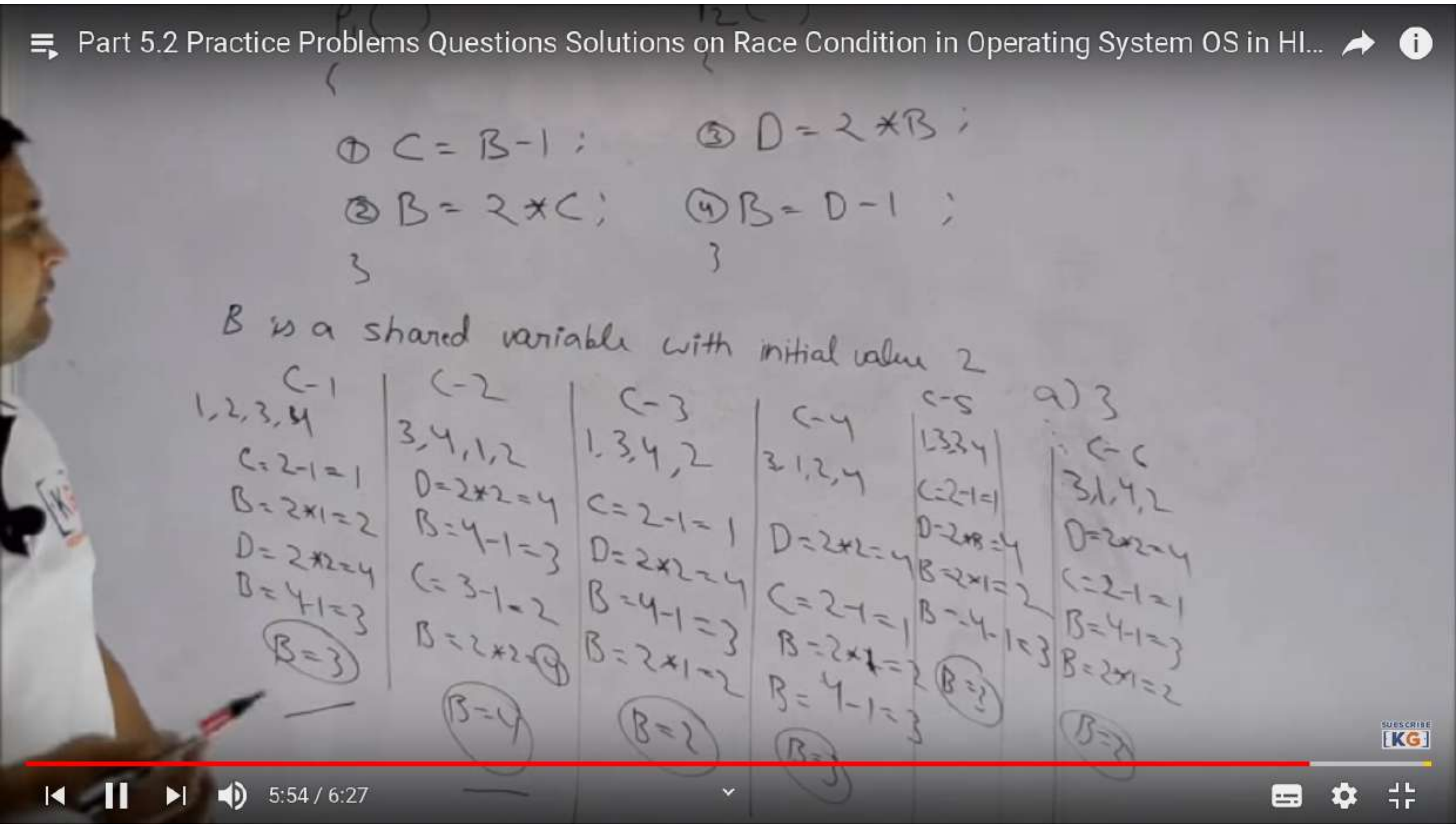
ii) Allow 4 Philosopher to Sit

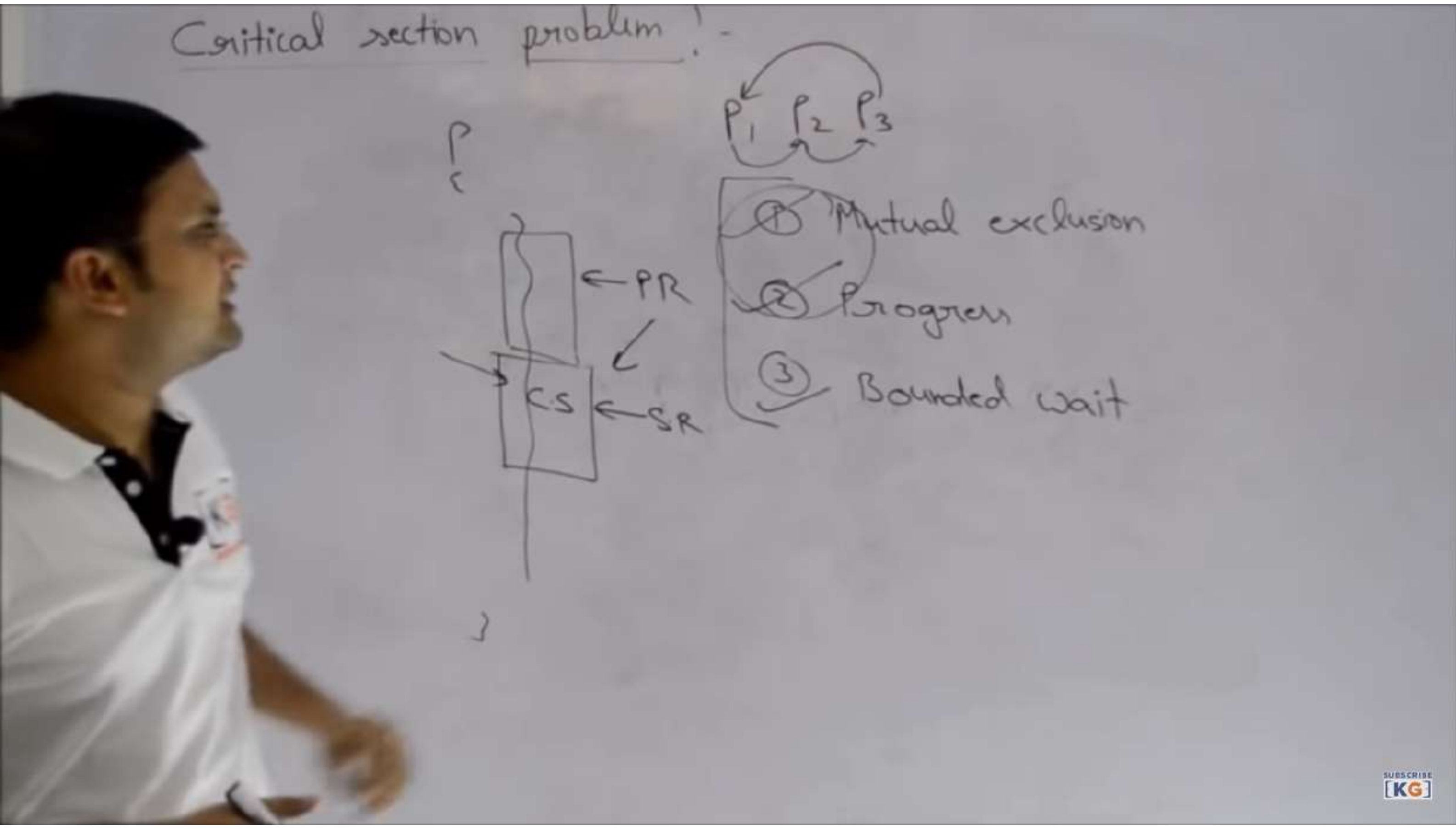
```
while(true)
{ Wait (Chopstick[i];
  wait (Chopstick[(i+1)%5];
   // Eat
  Signal (Shopstick[i]);
  Signal (chopstick [(i+1)%5];
}
```

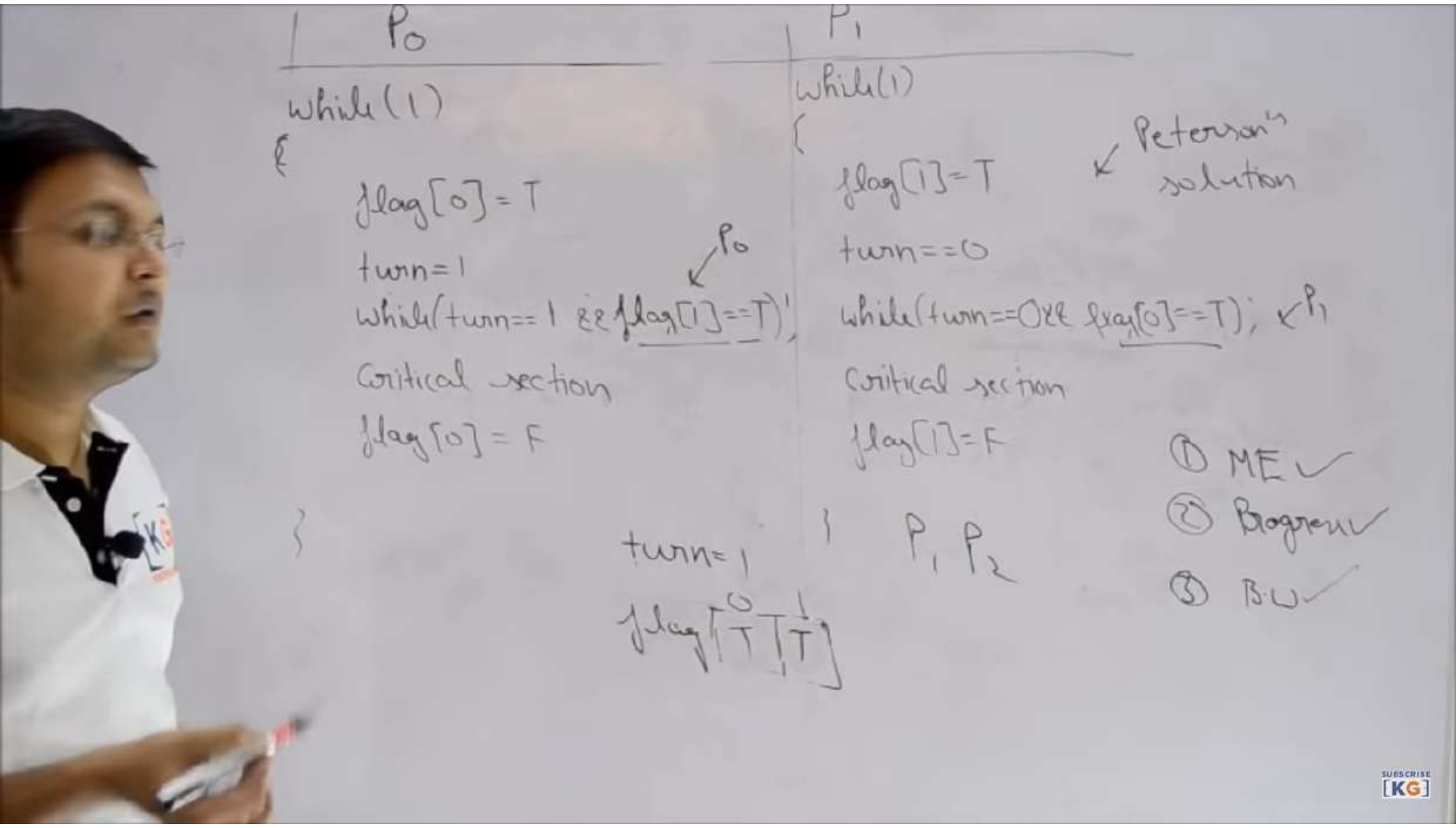


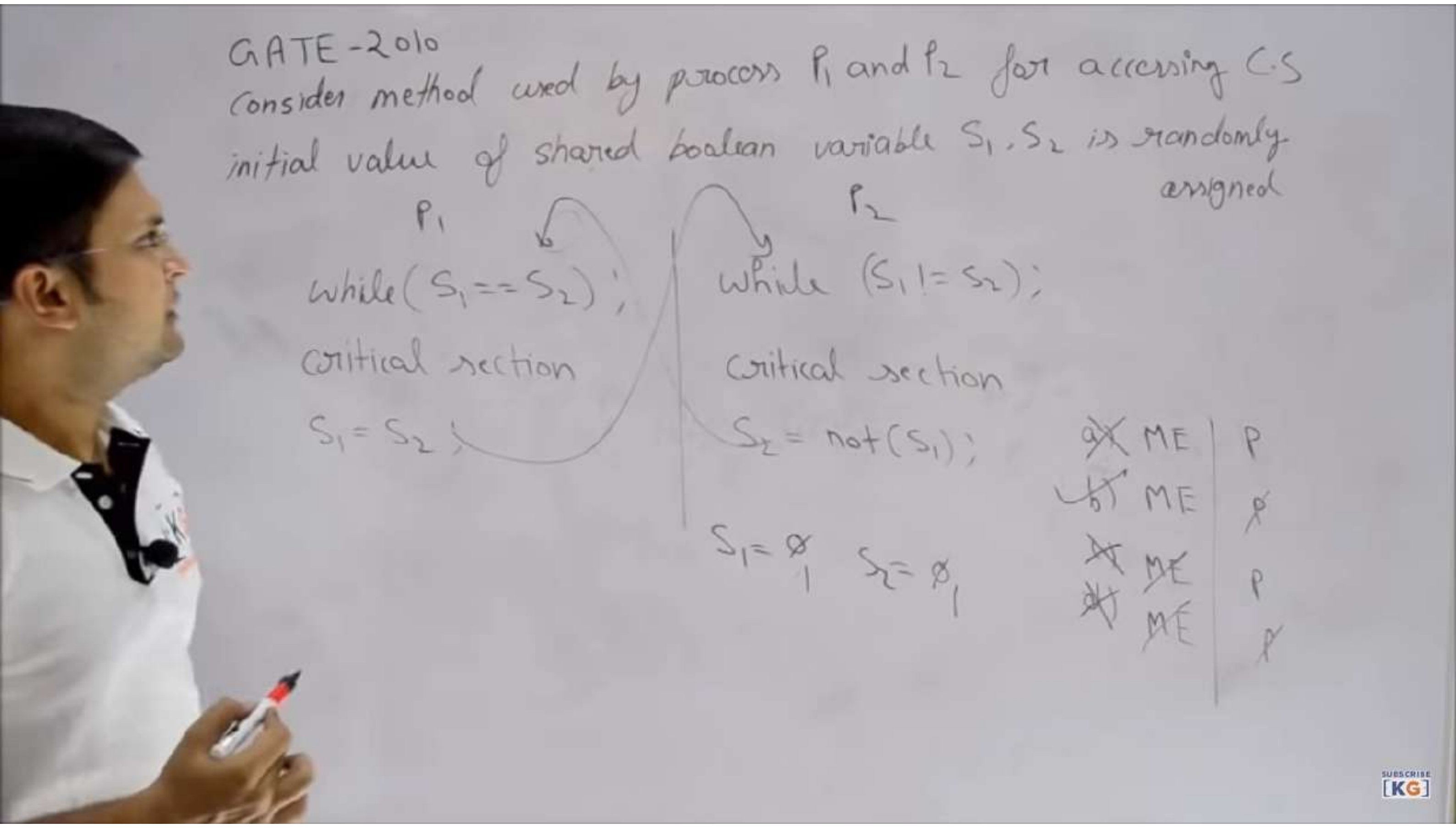
Synchronization mechanism.
4 Conditions / Rules.
Primary
1) Mutual Exclusion
2) Progress
Secondary
3) Bounded Wait
4) No assumption related to H/w Speed.
P1
P2
# include
main( )
main( )
A, B
Non critical Section
Entry Section
Count ++
CS
Exit Section
Non Critical
X, y.
Entry Section
Count - -
CS
Exit Section
SUBSCRIBE

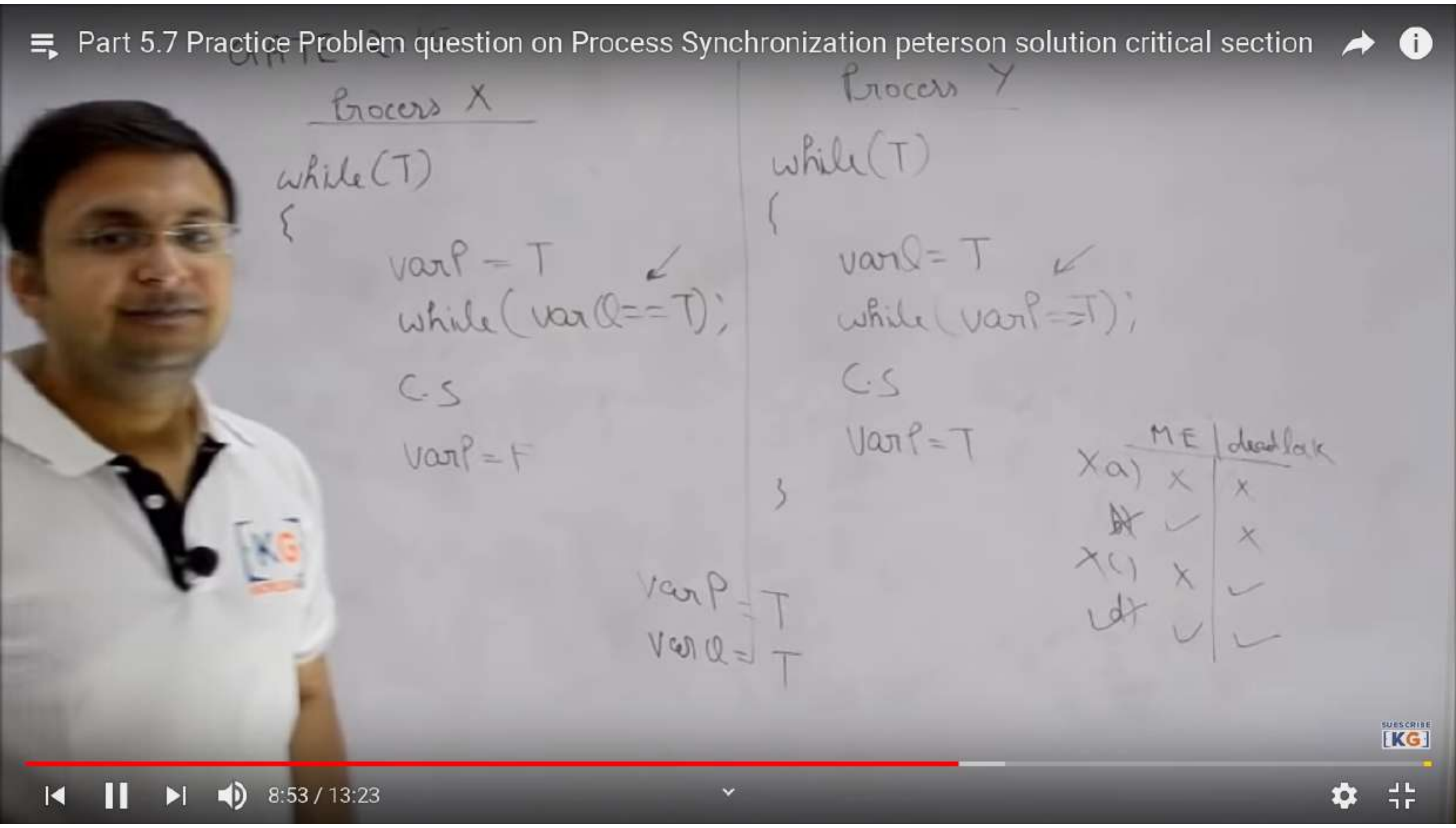

Void Consumer (Void)
{
int itemC;
While (true)
{
Out
Ø 1
Buffer Empty
While ((count == 0);
itemC = Buffer (out);
1. load Rc, m[Count]
2. DECR Rc
3. Store m[Count], Rc
Out = (Out+1) mod n;
Count = Count − 1;
Process_item (itemC);
}
}
(0+1) mod 8
1 mod 8 = 1
n=8
Buffer [0 .... n−1]
0
1
2
3
4
5
6
7
Case I. x1
(0+1) mod 8
1 mod 8 = 1
in
Ø 1
Count
Ø 1 0
int Count=0;
Void Producer (Void)
{
int itemp;
While (true)
{
Produce_item (itemp);
While (Count == n);
Buffer full
Buffer [in] = itemp;
in = (in+1) mod n;
Count = Count + 1
}
}
SUBSCRIBE


Part 5.2 Practice Problems Questions Solutions on Race Condition in Operating System OS in HI...
① C = B-1 ;
② B = 2*C ;
③ D = 2*B ;
④ B = D-1 ;
B is a shared variable with initial value 2
C-1
1,2,3,4
C=2-1=1
B=2*1=2
D=2*2=4
B=4-1=3
B=3
C-2
3,4,1,2
D=2*2=4
B=4-1=3
C=3-1=2
B=2*2=4
B=4
C-3
1,3,4,2
C=2-1=1
D=2*2=4
B=4-1=3
B=2*1=2
B=2
C-4
3,1,2,4
D=2*2=4
C=2-1=1
B=2*1=2
B=4-1=3
B=3
C-5
1,3,2,4
C=2-1=1
D=2*B=4
B=2*1=2
B=4-1=3
B=3
a) 3
C-6
3,1,4,2
D=2*2=4
C=2-1=1
B=4-1=3
B=2*1=2
B=2
5:54 / 6:27


Critical section problem :-
P1 P2 P3
P
{
PR
C.S
SR
}
1 Mutual exclusion
2 Progress
3 Bounded wait


P0
P1
while(1)
{
flag[0] = T
turn=1
P0
while(turn==1 && flag[1]==T);
critical section
flag[0] = F
}
while(1)
{
flag[1]=T
Peterson's solution
turn==0
while(turn==0 && flag[0]==T);
P1
critical section
flag[1]=F
}
P1 P2
turn=1
flag[T T]
0 1
① ME
② Progress
③ B.W
KG


GATE-2010
Consider method used by process P1 and P2 for accessing C.S
initial value of shared boolean variable S1, S2 is randomly assigned
P1
while(S1==S2);
critical section
S1=S2;
P2
while (S1!=S2);
critical section
S2 = not(S1);
S1= 0 1   S2= 0 1
a) ME | P
b) ME | P
ME | P
ME | P
SUBSCRIBE KG


Part 5.7 Practice Problem question on Process Synchronization peterson solution critical section
Process X
while(T)
{
varP = T
while(varQ==T);
C.S
varP = F
Process Y
while(T)
{
varQ = T
while(varP==T);
C.S
VarP = T
}
varP = T
VarQ = T
M.E | deadlock
8:53 / 13:23
SUBSCRIBE
KG