# Assignment# 02

# Name: M WASEEM Reg# FA16-BSE-171(5C)

**Dekker’sAlgorithm**

**Question:- Understand and implement all the version of Dekker’s Algorithm for handing the concurrency issue**.

**First version of Dekker’salgorithm :**

(1) Succeeds in enforcing mutual exclusion .

(2) Uses variable to control which thread can execute.

(3) Constantly tests whether critical section is available .

• Busy waiting

• Wastes significant processor time

(4) Problem known as lockstep synchronization.

• Each thread can execute only in strict alternation.

**Implementation:**

System:

Int threadNumber=1;

startThreads();

TreadT1:

Void main(){

While(!done){

While(threadNumber==2);

//Critical Section

threadNumber==2;

}

} //end Thread1

ThreadT2:

Void main(){

While(!done){

While(threadNumber==1);

//Critical Section

threadNumber==1;

}

}

**Second version** **:**

(1) Removes lockstep synchronization.

(2) Violates mutual exclusion .

•Thread could be preempted while updating flag variable.

(3) Not an appropriate solution

**Implementation:**

System:

boolean t1Inside=false;

boolean t1Inside=false;

startThreads();

TreadT1:

Void main(){

While(!done){

While(t2Inside);

t1Inside=true;

//Critical Section

t1Inside=false;

}

} //end Thread1

ThreadT2:

Void main(){

While(!done){

While(t1Inside);

t2Inside=true;

//Critical Section

t2Inside=false;

}

}

**Third version :**

(1) Set critical section flag before entering critical section test.

•Once again guarantees mutual exclusion

(2) Introduces possibility of deadlock .

•Both threads could set flag simultaneously.

•Neither would ever be able to break out of loop.

(3) Not a solution to the mutual exclusion problem.

**Implementation:**

System:

boolean t1WantsToEnter=false;

boolean t2WantsToEnter =false;

startThreads();

TreadT1:

Void main(){

While(!done){

t1WantsToEnter=true;

While(t2WantsToEnter);

//Critical Section

t1WantsToEnter=false;

}

} //end Thread1

ThreadT2:

Void main(){

While(!done){

t2WantsToEnter=true;

While(t1WantsToEnter);

//Critical Section

t2WantsToEnter=false;

}

}

**Fourth version :**

(1) Sets flag to false for small periods of time to yield control .

(2) Solves previous problems, introduces indefinite postponement.

• Both threads could set flags to same values at same time

•Would require both threads to execute in tandem (unlikely but possible)

(3) Unacceptable in mission-or business-critical system.

**Implementation:**

Trying[0] = TRUE;    
while (Trying[1] == TRUE)  {  
  if (FavoredP == 1)    
  {  
    Trying[0] = FALSE    
    while (Trying[1] == TRUE);    
    Trying[0] = TRUE;    
  }  
}    
  
criticalSection(0);        
FavoredP = 1;              
Trying[0] = FALSE;

Trying[1] = TRUE;    
while (Trying[0] == TRUE)   
  if (FavoredP == 0)    
  {  
    Trying[1] = FALSE     while (Trying[0] == TRUE);    
    Trying[1] = TRUE;    
  }  
}    
  
criticalSection(1);        
FavoredP = 0;              
Trying[1] = FALSE;