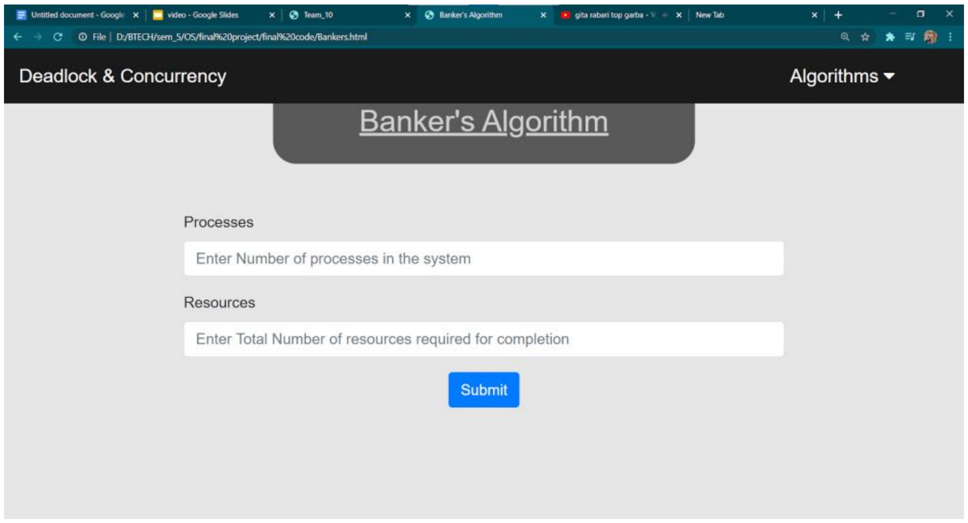
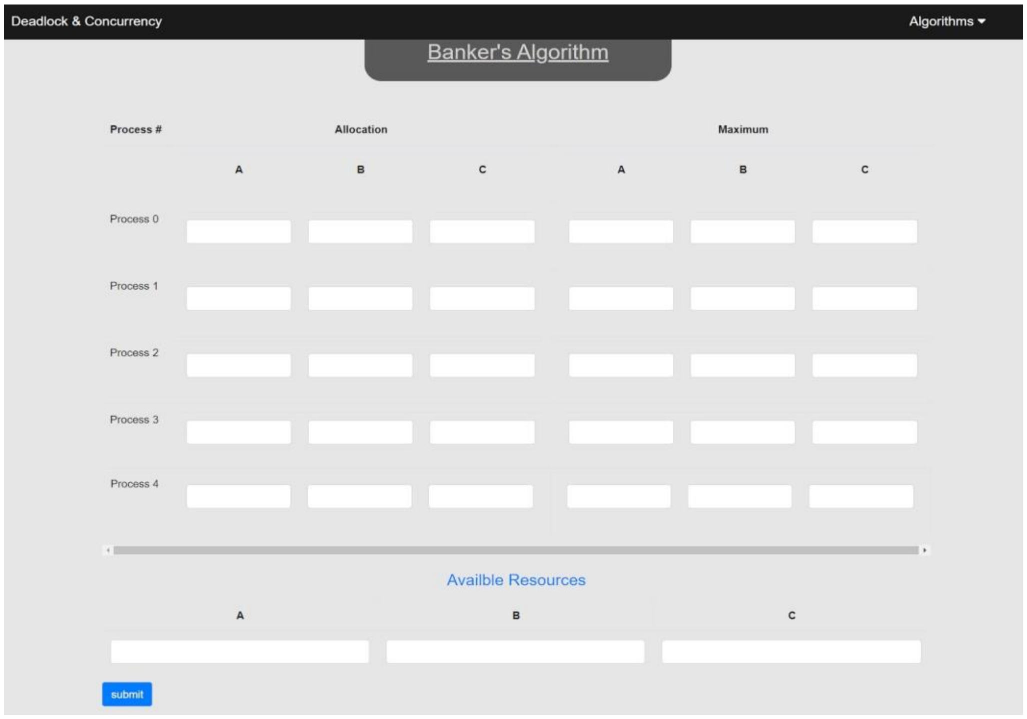
1.Banker’s Algorithm

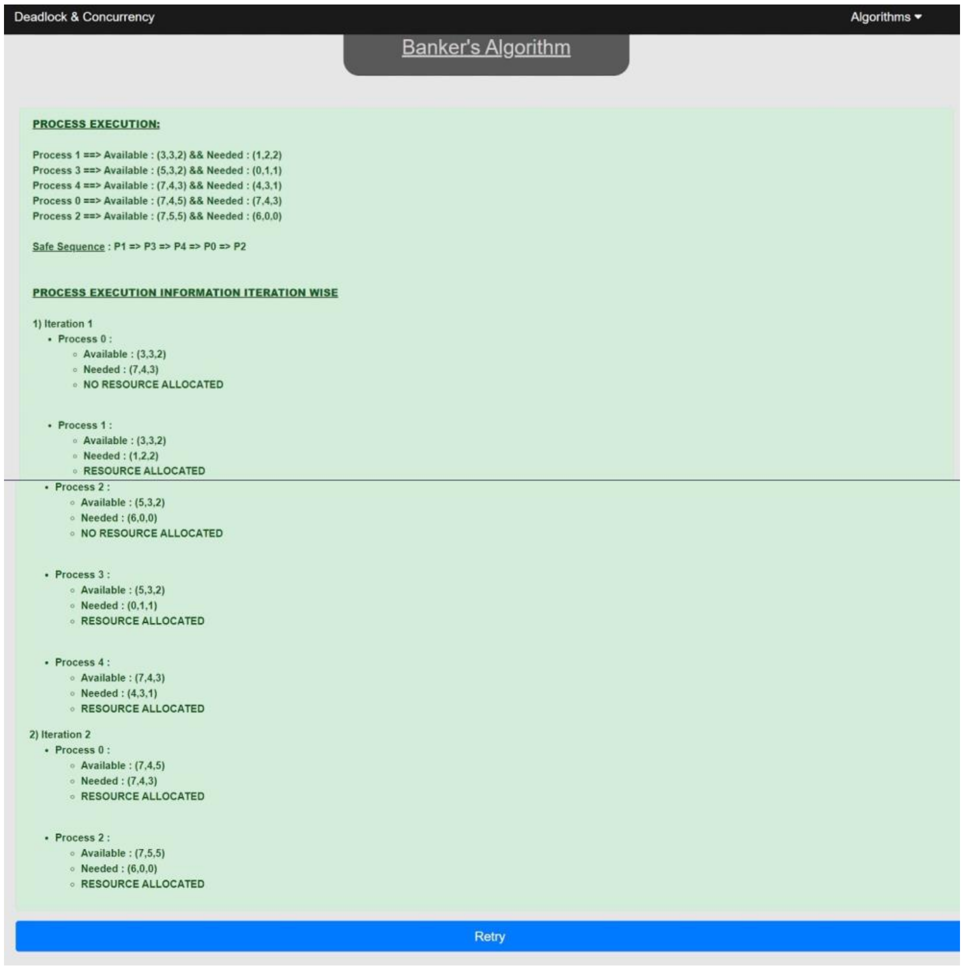
- The banker's algorithm is a resource allocation algorithm and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources and making decisions whether allocation should be allowed or not.


Deadlock & Concurrency
Algorithms
Banker's Algorithm
Processes
Enter Number of processes in the system
Resources
Enter Total Number of resources required for completion
Submit

- Entre Number of Process and Resources to Proceed



- Then have to enter Allocation & Maximum Resources of each process.
- Also need to enter Available Resources. Click Submit to Proceed.

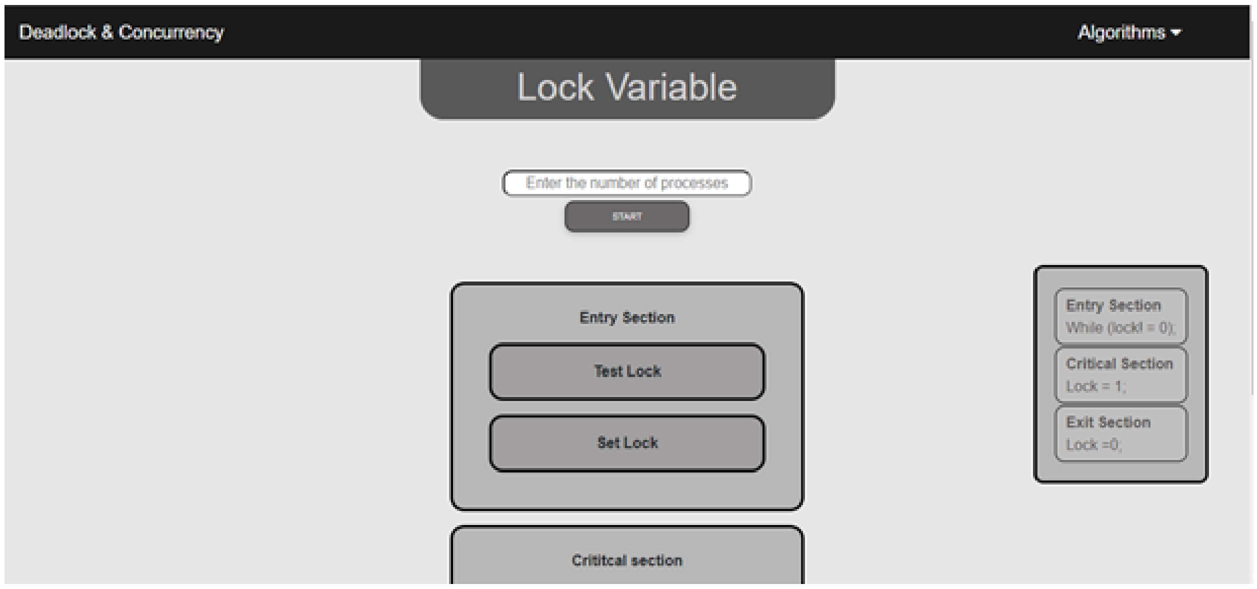


```
PROCESS EXECUTION:

Process 1 ==> Available : (3,3,2) && Needed : (1,2,2)
Process 3 ==> Available : (5,3,2) && Needed : (0,1,1)
Process 4 ==> Available : (7,4,3) && Needed : (4,3,1)
Process 0 ==> Available : (7,4,5) && Needed : (7,4,3)
Process 2 ==> Available : (7,5,5) && Needed : (6,0,0)

Safe Sequence : P1 => P3 => P4 => P0 => P2


PROCESS EXECUTION INFORMATION ITERATION WISE

1) Iteration 1
  • Process 0 :
      ◦ Available : (3,3,2)
      ◦ Needed : (7,4,3)
      ◦ NO RESOURCE ALLOCATED

  • Process 1 :
      ◦ Available : (3,3,2)
      ◦ Needed : (1,2,2)
      ◦ RESOURCE ALLOCATED
  • Process 2 :
      ◦ Available : (5,3,2)
      ◦ Needed : (6,0,0)
      ◦ NO RESOURCE ALLOCATED

  • Process 3 :
      ◦ Available : (5,3,2)
      ◦ Needed : (0,1,1)
      ◦ RESOURCE ALLOCATED

  • Process 4 :
      ◦ Available : (7,4,3)
      ◦ Needed : (4,3,1)
      ◦ RESOURCE ALLOCATED
2) Iteration 2
  • Process 0 :
      ◦ Available : (7,4,5)
      ◦ Needed : (7,4,3)
      ◦ RESOURCE ALLOCATED

  • Process 2 :
      ◦ Available : (7,5,5)
      ◦ Needed : (6,0,0)
      ◦ RESOURCE ALLOCATED
```

- After clicking on submit the output will be shown.
- In output it will display Safe sequence of process execution.
- It will also display Needed Resources of each process.

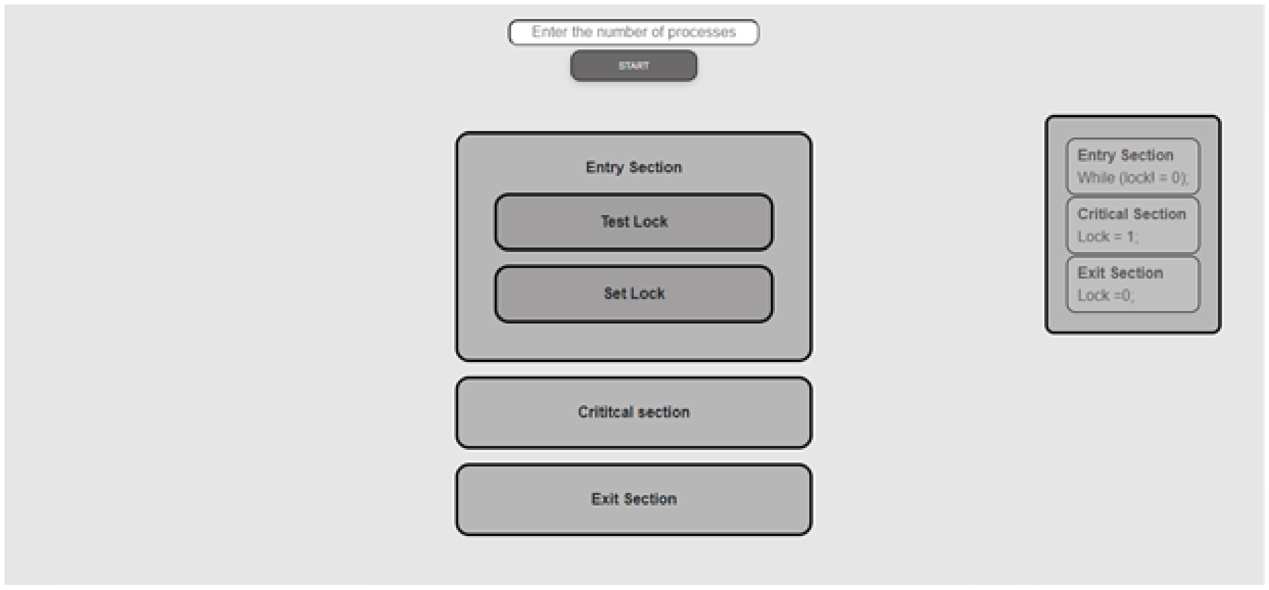
2.Lock Variable

- This is the simplest synchronization mechanism. This is a Software Mechanism implemented in User mode. This is a busy waiting solution which can be used for more than two processes. In this mechanism, a Lock variable **lock** is used. Two values of lock can be possible, either 0 or 1. Lock value 0 means that the critical section is vacant while the lock value 1 means that it is occupied. A process which wants to get into the critical section first checks the value of the lock variable. If it is 0 then it sets the value of lock as 1 and enters into the critical section, otherwise it waits.



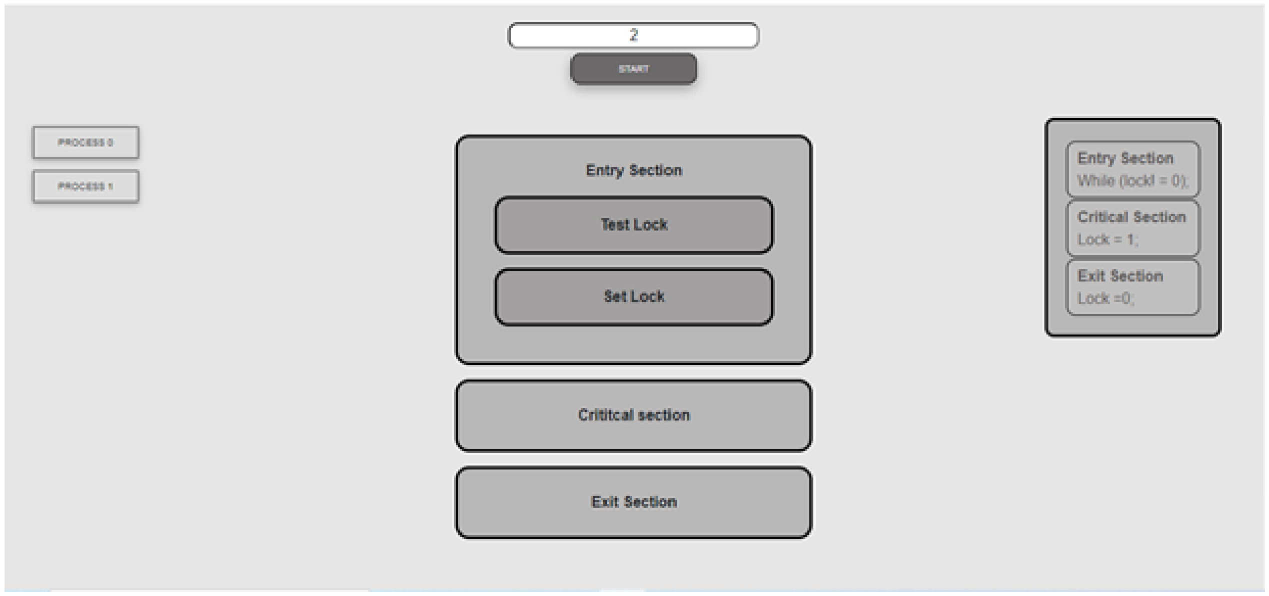
Entry Section
While (lockl = 0);
Critical Section
Lock = 1;
Exit Section
Lock =0;

- Enter the number of processes to proceed.

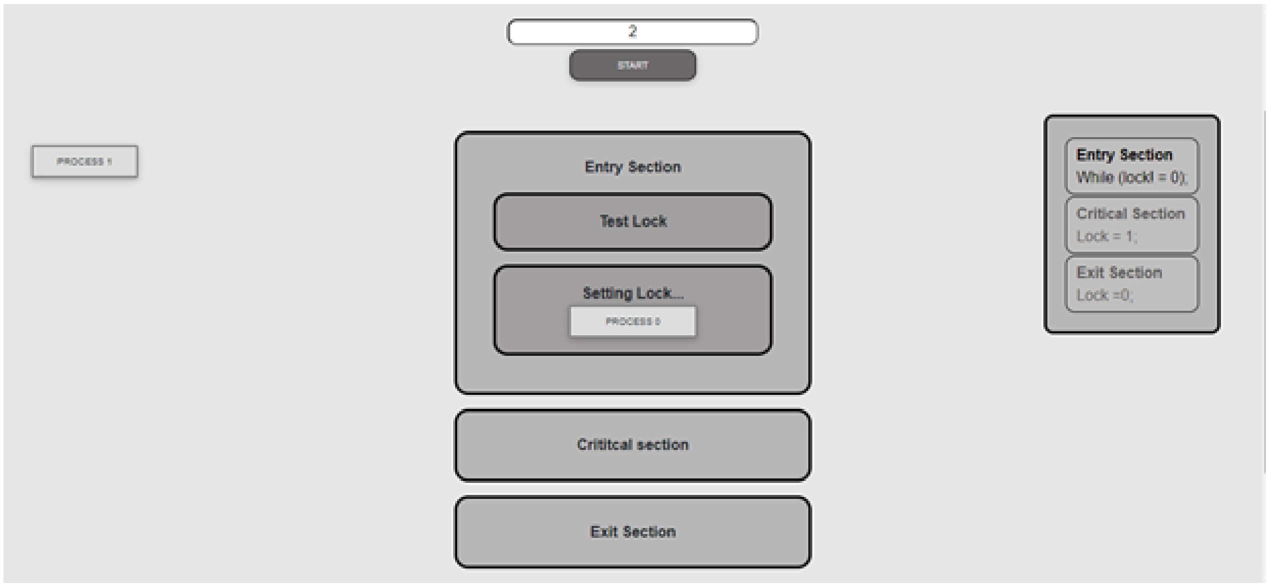


Entry Section
While (lockl = 0);
Critical Section
Lock = 1;
Exit Section
Lock =0;

- After entering the processes, you need to select the process.



- Then the selected process goes through all the section step by step.

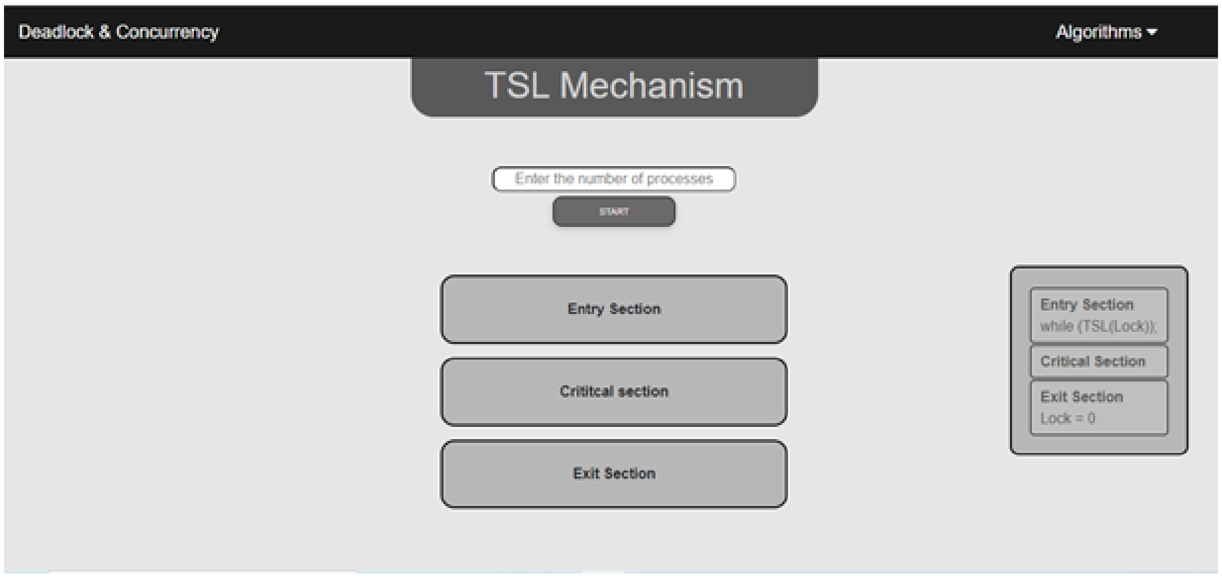


- In mean time, the other process has to wait till the first selected process reaches to exit section.
- Then the same process will be follow up for the rest of the processes.

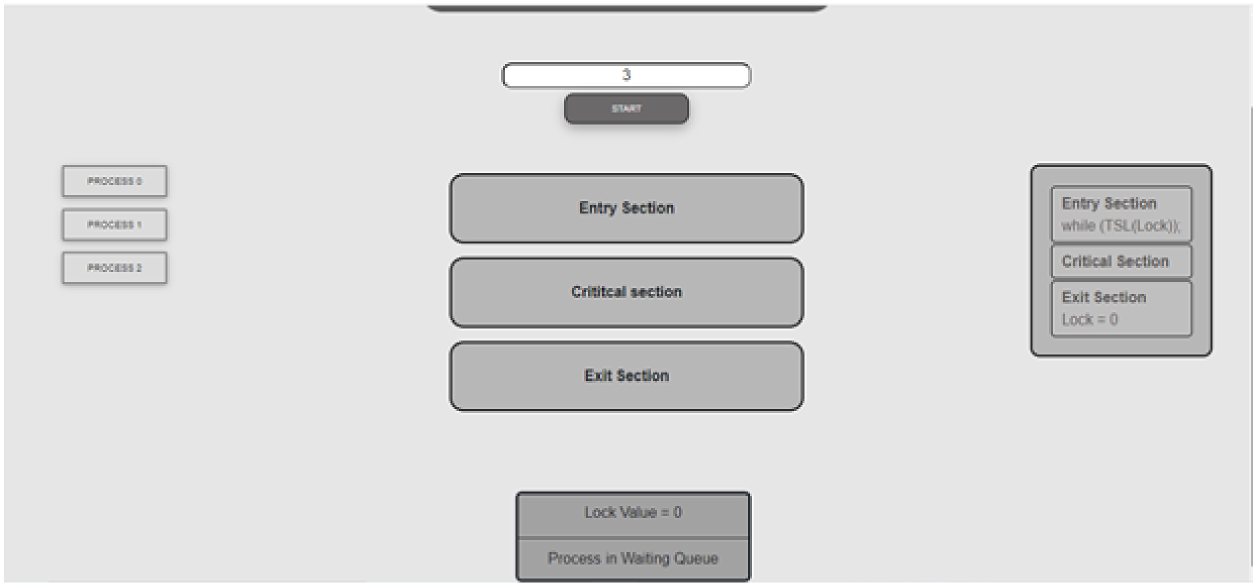
3.Test Set Lock

- Test and Set Lock (TSL) is a synchronization mechanism. It uses a test and set instruction to provide the synchronization among the processes executing concurrently.

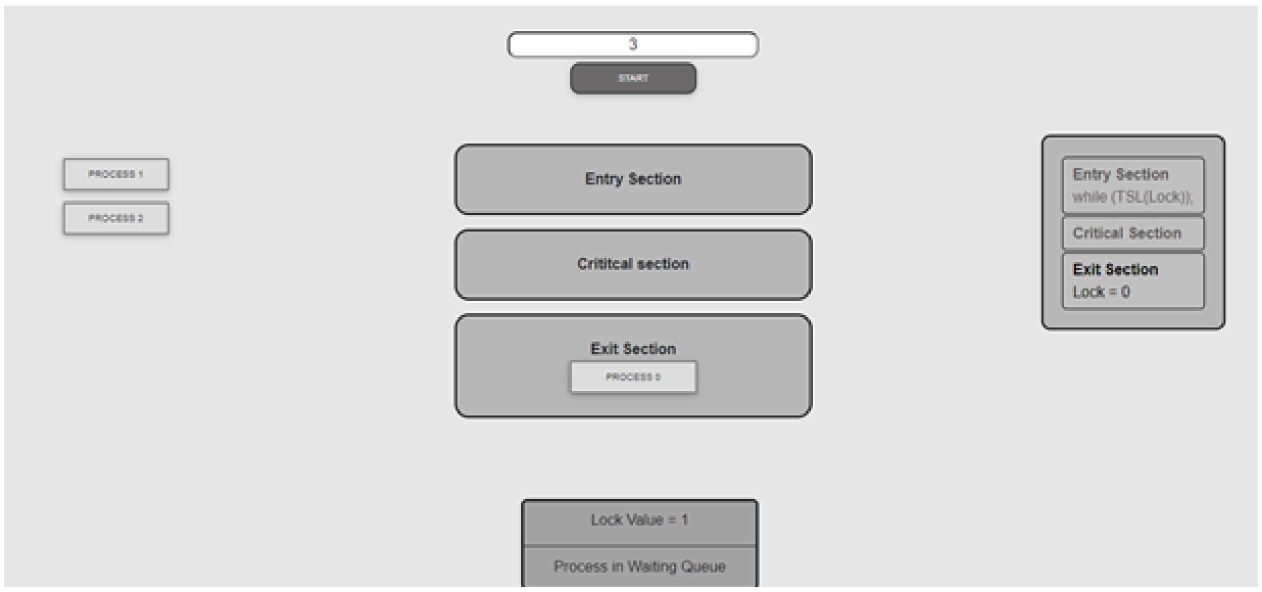
  It is an instruction that returns the old value of a memory location and sets the memory location value to 1 as a single atomic operation. If one process is currently executing a test-and-set, no other process is allowed to begin another test-and-set until the first process test-and-set is finished.



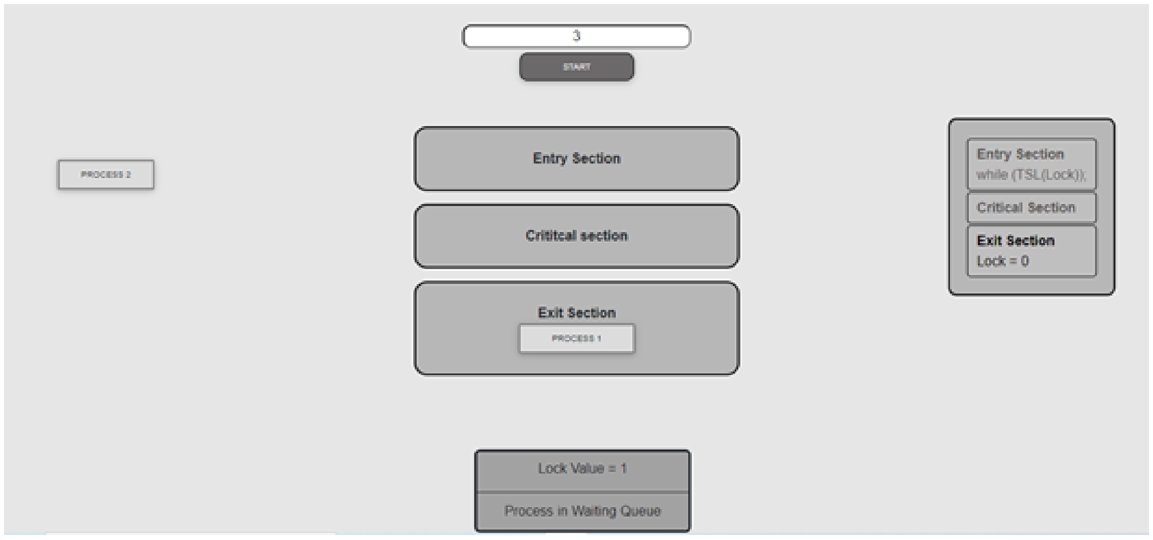
- Enter the number of processes to proceed.
- Click on the start button after entering.



- Select the process from queue.
- It will go through all the sections and enable other processes to proceed until it reaches exit section.
- The value of the Lock will be 1 while the selected process is processing through the sections.

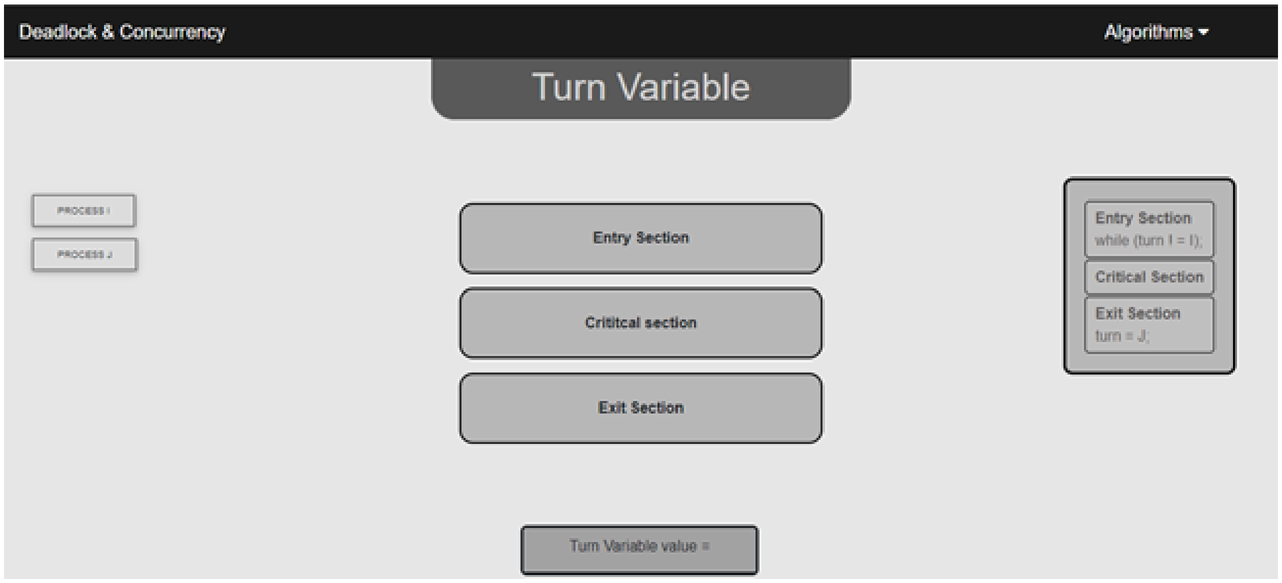


- The above-mentioned steps will be followed till we reach the last process in the queue.

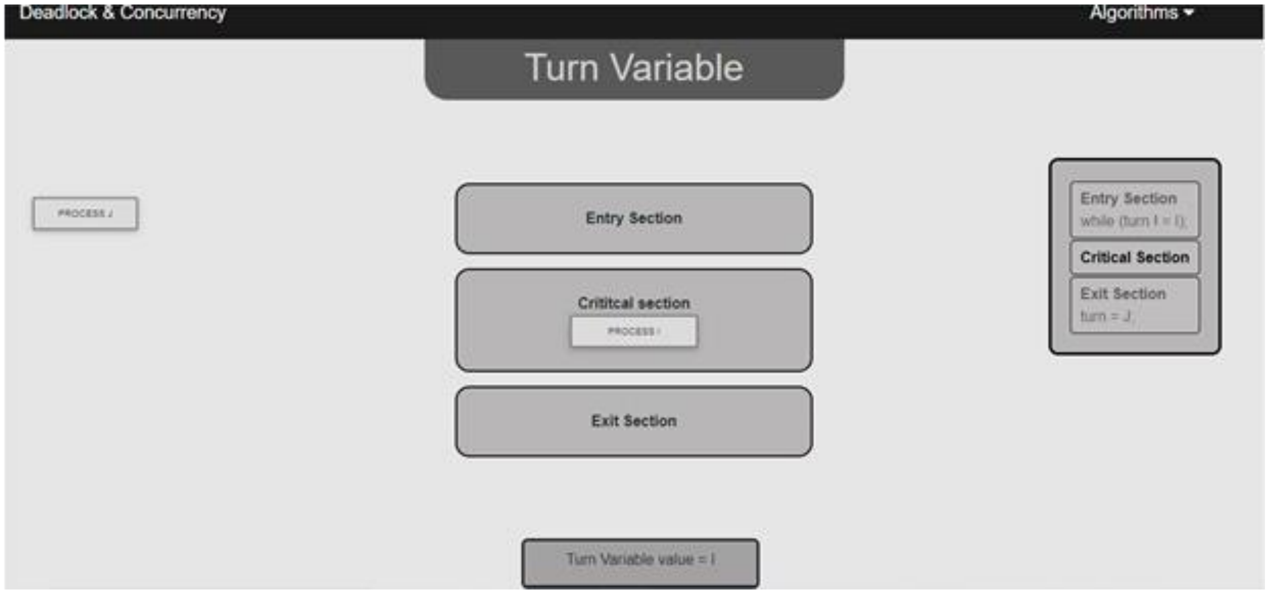


4.Turn Variable

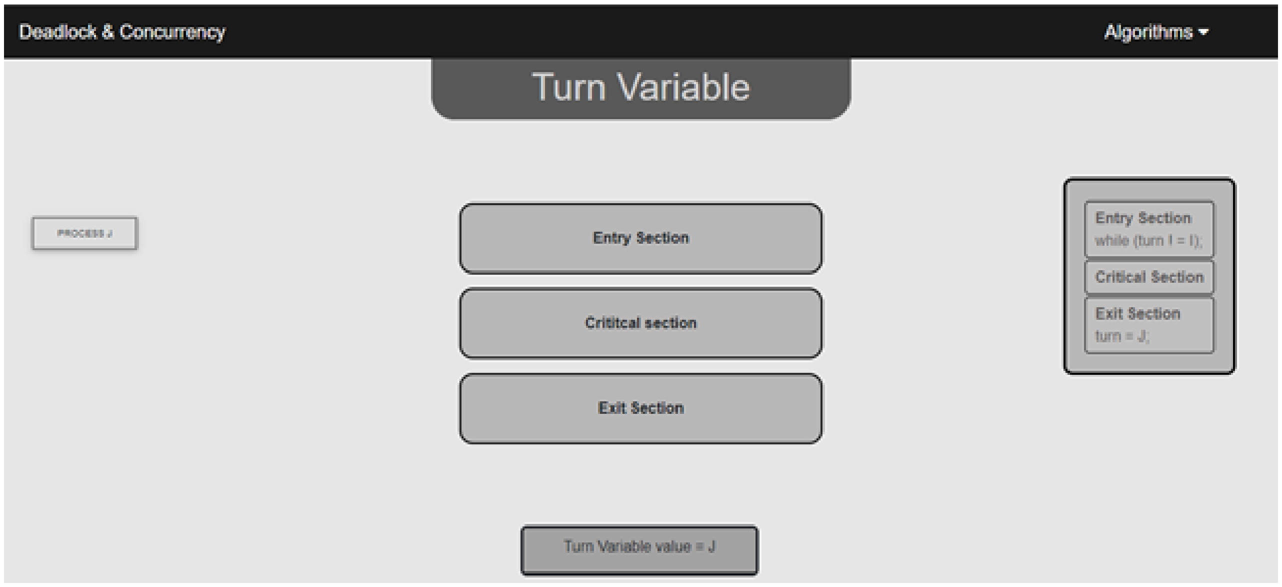
- **Turn Variable** or Strict Alternation Approach is the software mechanism implemented at user mode. It is a busy waiting solution which can be implemented only for two processes. In this approach, A **turn variable** is used which is actually a lock. This approach can only be used for only two processes.



- You can select either process at a time.
- The Turn Variable Value will hold the value of the process which is running through the sections.



- As soon as the first process comes to exit section, the value of Turn Variable will be changed to next process in the queue.



```
Entry Section
while (turn ! = I);
Critical Section
Exit Section
turn = J;
```

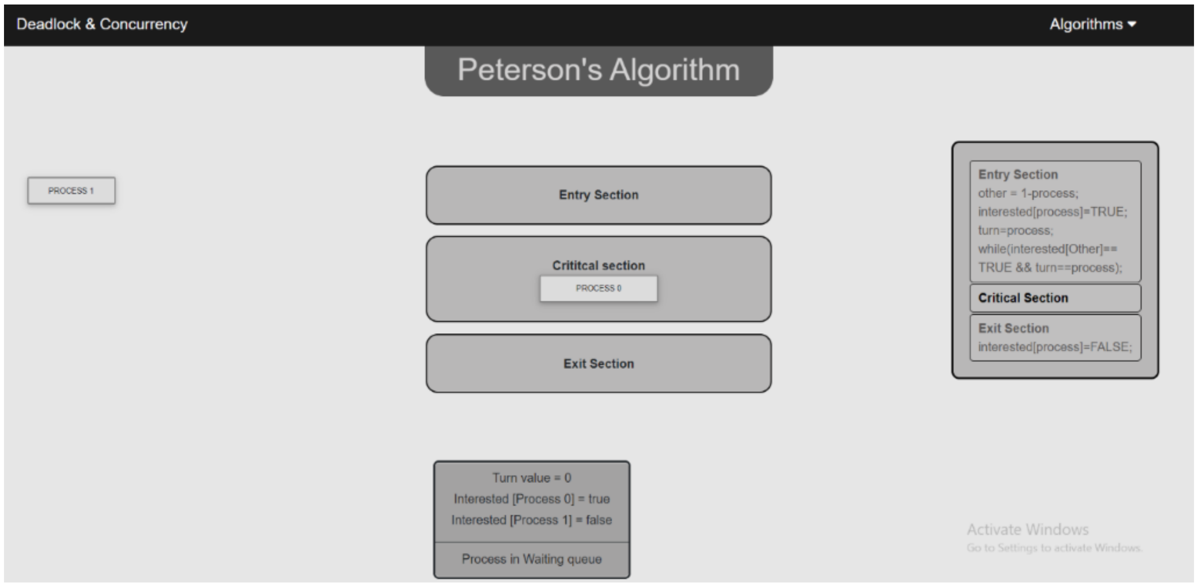
- All the steps mentioned above will be followed by all the processes in the queue.

5.Peterson Method

- Combined shared variables of algorithms 1 and 2.
- Process Pi

```
do   {
        flag [i]:= true;
        turn = j;
        while (flag [j] and turn = j) do nothing;
        critical section flag [i] = false;
        remainder section
 } while (1);
```

- Meets all three requirements; solves the critical-section problem for two processes.

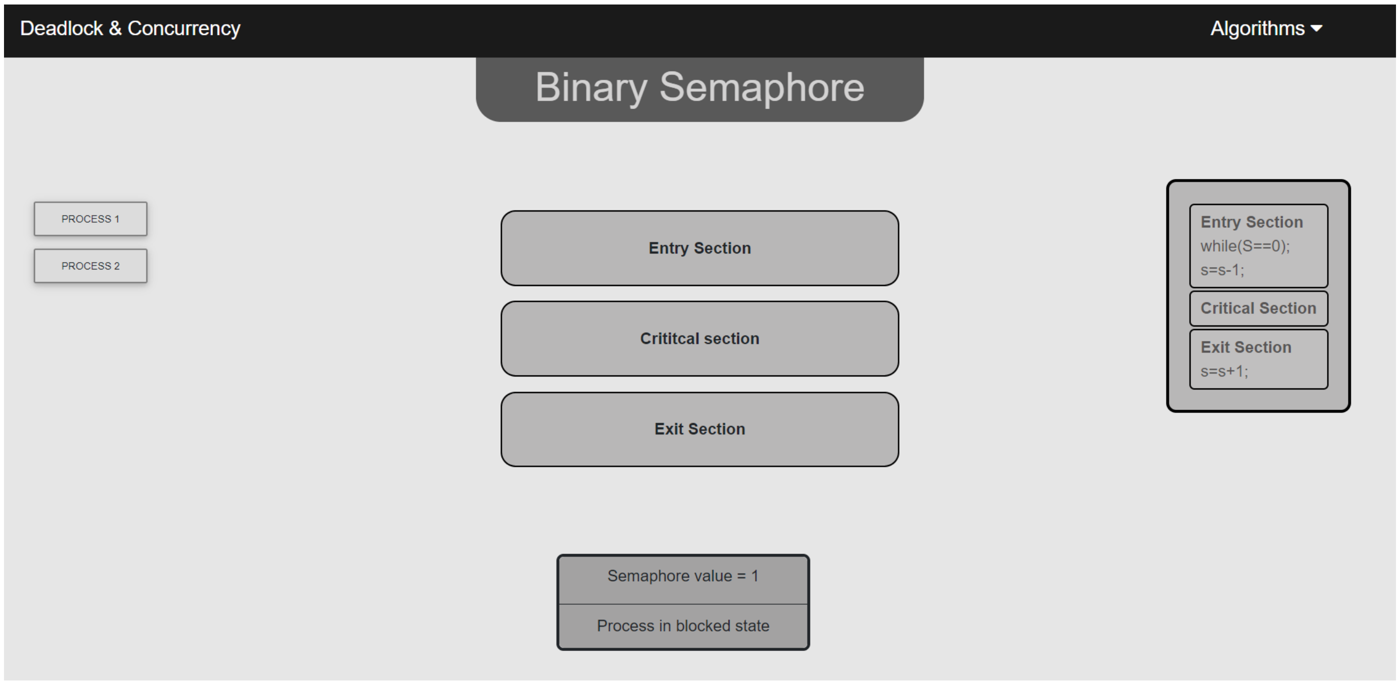


```
Entry Section
other = 1-process;
interested[process]=TRUE;
turn=process;
while(interested[Other]==
TRUE && turn==process);
Critical Section
Exit Section
interested[process]=FALSE;
```

```
Turn value = 0
Interested [Process 0] = true
Interested [Process 1] = false
Process in Waiting queue
```

6.Binary Semaphore

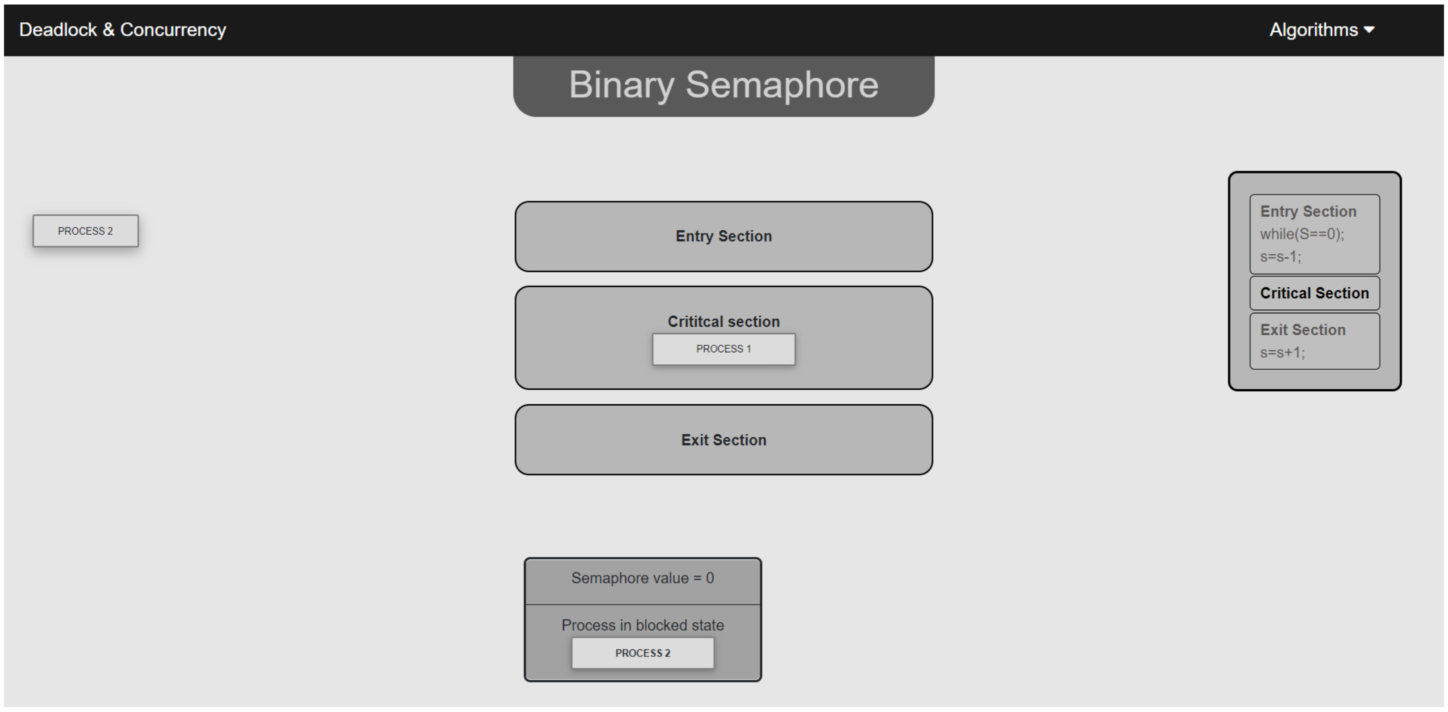
- They are also known as mutex locks as it provides mutual exclusion. This type of semaphores have their value restricted to 0 and 1. The wait operation works only if semaphore=1, and the signal operation succeeds when semaphore=0.



Entry Section
while(S==0);
s=s-1;
Critical Section
Exit Section
s=s+1;

Semaphore value = 1
Process in blocked state

- In binary semaphore two process are available. Process 1 and Process 2.
- When clicking on any process like Process 1 or Process 2 it checks the semaphore value if semaphore value is 1 then it goes into entry section otherwise it will wait in blocked state until semaphore value becomes 1.
- First Process goes into the Entry Section. Then it goes into Critical section and execute it. Then at last it goes into Exit Section to terminate the Process and then Semaphore value becomes 1.



```
while(S==0);
s=s-1;
```

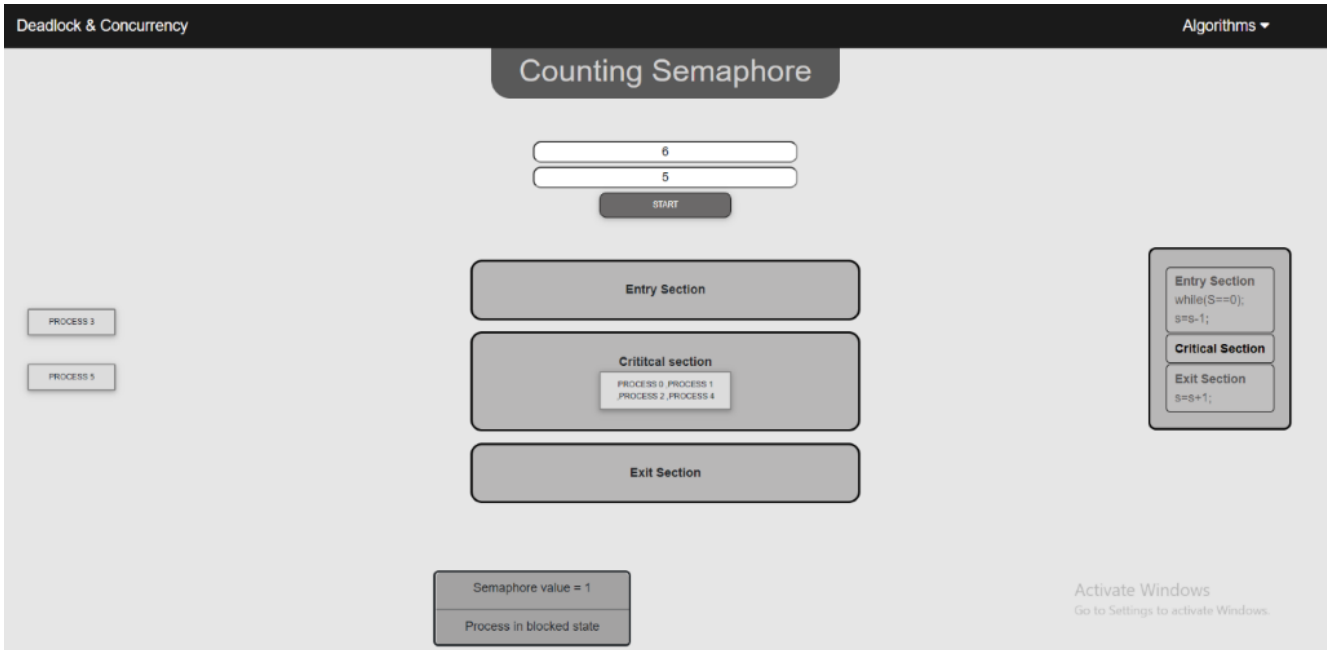
```
s=s+1;
```

Semaphore value = 0

- In the above snapshot Process 1 is in the Critical Section and also Semaphore value is 0.
- Also, the Process 2 is in the blocked state.
- Whenever Process 1 finishes executing then semaphore becomes 1 and Process 2 starts executing.

7.Counting Semaphore

- Semaphore is a variable that has an integer value
- May be initialized to a nonnegative number
- Wait operation decrements the semaphore value
- Wait and signal operations cannot be interrupted
- If a process is waiting for a signal, it is suspended until that signal is sent
- Queue is used to hold processes waiting on the semaphore



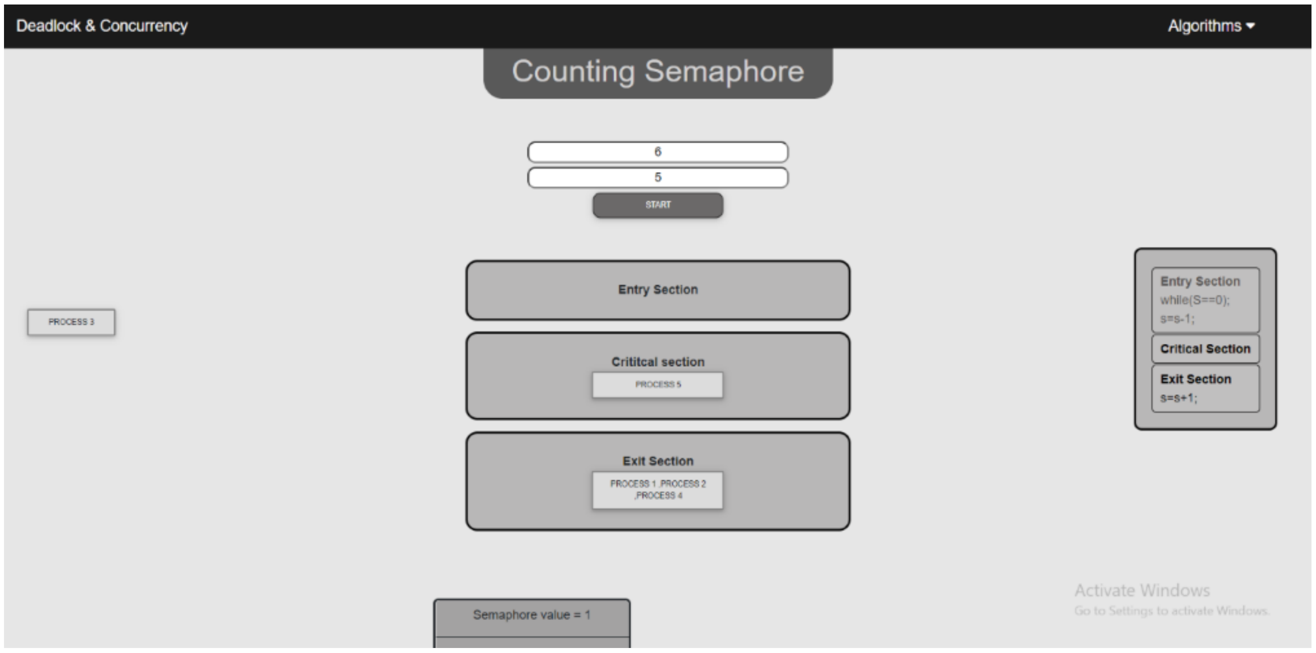
6
5

PROCESS 0 ,PROCESS 1 ,PROCESS 2 ,PROCESS 4

```
Entry Section
while(S==0);
s=s-1;
Critical Section
Exit Section
s=s+1;
```

Semaphore value = 1

Process in blocked state



6
5

PROCESS 5

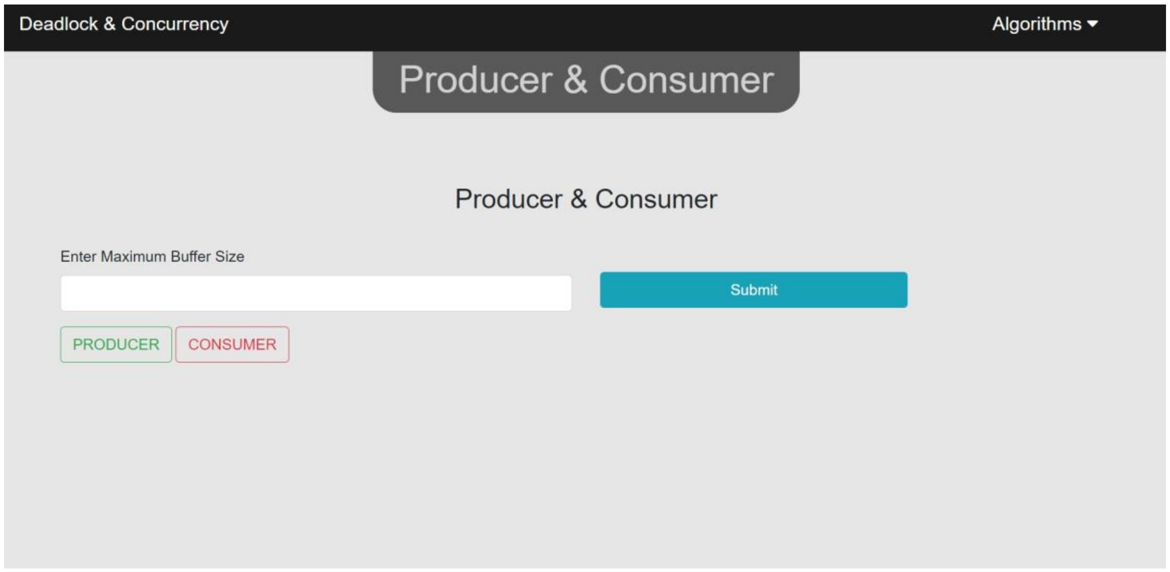
PROCESS 1 ,PROCESS 2 ,PROCESS 4

```
Entry Section
while(S==0);
s=s-1;
Critical Section
Exit Section
s=s+1;
```

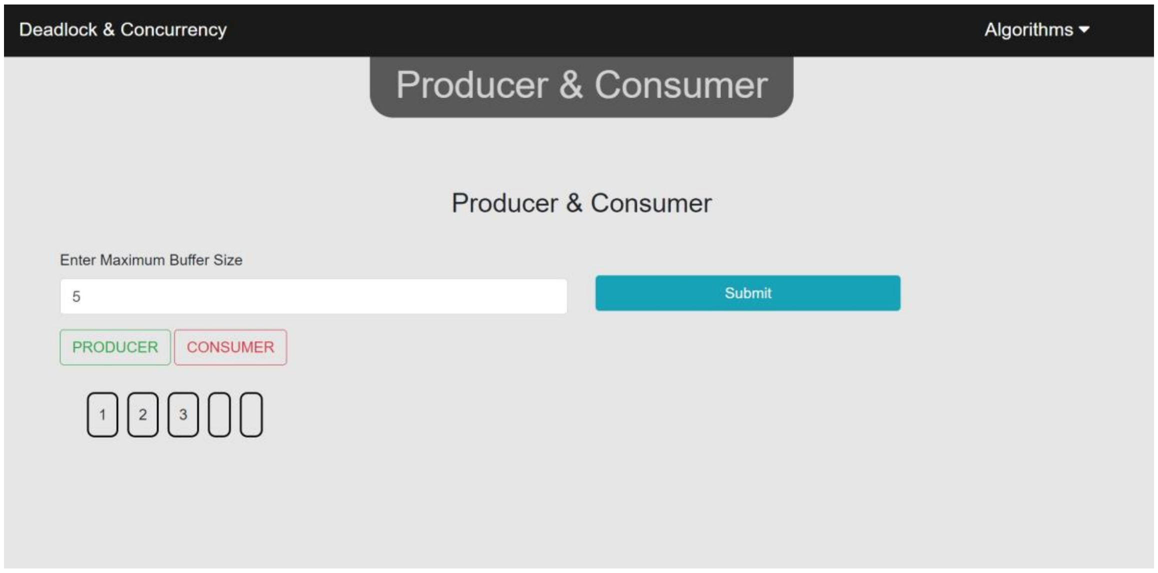
Semaphore value = 1

8.Producer and Consumer

- It is a classical multi-process synchronization problem. There is one producer and one consumer. Producer is producing some items until buffer is full, whereas consumer consumes the item until buffer is empty. They share same memory buffer which is of fixed size. But both cannot access the memory buffer at the same time.



- Enter maximum buffer size to proceed.
- When you click on Producer if space is available then it will produce or else space is not available then it shows an error.
- When you click on Consumer if any item is there to consume then it consumes otherwise it shows an error.

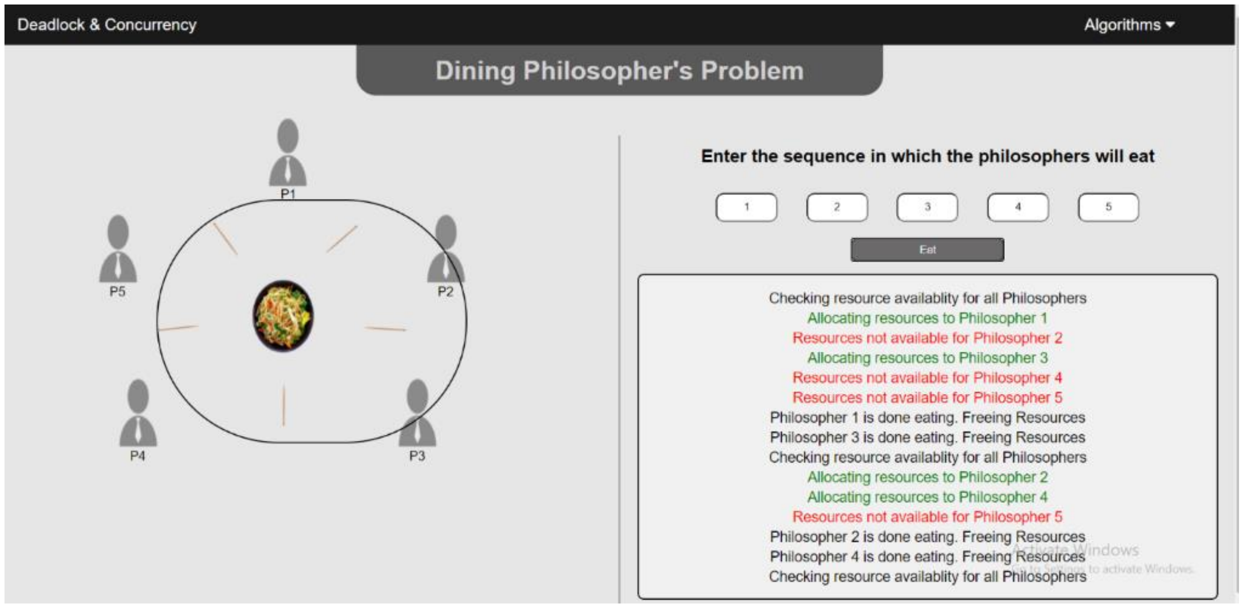


- In the above snapshot Buffer size set to 5 and three of them filled. So, consumer can consume it.

- When click on consumer button it will consume 1 item from buffer from starting node. If buffer is empty then shows an error.

9.Dining Philosopher's problem

- Initially all values are 1
- To start eating, a philosopher needs two chopsticks
- A philosopher may pick up only one chopstick at a time
- After eating, the philosopher releases both the chopsticks
- The solution is not deadlock free!!
- All philosophers pick up left chopsticks!!
- Allow at most 4 philosophers to be sitting on the table
- Allow a philosopher to pick chopsticks only if both are available
- An odd philosopher picks up first the left and then the right chopstick while a even philosopher does the reverse


Deadlock & Concurrency
Algorithms
Dining Philosopher's Problem
P1
P2
P3
P4
P5
Enter the sequence in which the philosophers will eat
1
2
3
4
5
Eat

```
Checking resource availablity for all Philosophers
Allocating resources to Philosopher 1
Resources not available for Philosopher 2
Allocating resources to Philosopher 3
Resources not available for Philosopher 4
Resources not available for Philosopher 5
Philosopher 1 is done eating. Freeing Resources
Philosopher 3 is done eating. Freeing Resources
Checking resource availablity for all Philosophers
Allocating resources to Philosopher 2
Allocating resources to Philosopher 4
Resources not available for Philosopher 5
Philosopher 2 is done eating. Freeing Resources
Philosopher 4 is done eating. Freeing Resources
Checking resource availablity for all Philosophers
```