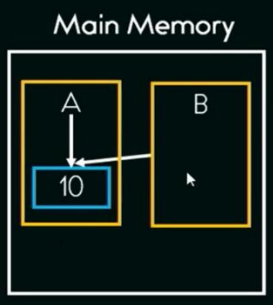
# Shared Memory Model

The shared memory model establishes a region of memory that is shared by cooperating processes.

- The processes exchange information by reading and writing data to the shared region.
- This model is faster than message passing model and provides maximum speed.
- Typically, the shared memory region resides in the address space of the process that creates this region.
- The other processes requiring to communicate using that shared memory region must attach to their address space.
- The operating system normally prevents one process to access the memory of another process.
- All processes wishing to communicate using this model must agree to remove this restriction.




0:03:25 0:07:47
21°C Cloudy 9:19 AM 10/7/2022

# Shared Memory Model

(Producer Consumer Example)

This is very simple example of two cooperating processes.

- These two processes are producer and consumer.

**Producer Process:** It produces information that will be consumed by the consumer.

**Consumer Process:** It consumes information produced by the producer.

- Both processes run concurrently.
- If the consumer has nothing to consume, it waits.

There are two versions of the producer.

- In version one, the producer can produce an infinite amount of items. This is called unbounded buffer producer consumer problem.
- In the other version, there is a fixed limit to the buffer size. This is called bounded buffer producer consumer problem.


0:03:25 0:07:47
21°C Cloudy 9:20 AM 10/7/2022

# Shared Memory Model

(Producer Consumer Example)

## Bounded Buffer – Shared Memory Solution

```
//shared data:
   int n=5, item, in ,out;
   int buffer[n];
//both the consumer and producer are
currently looking at buffer element 1.
   In = 1;      out = 1;
//Producer Process:
While(true)
{  while ( in+1 % n == out )
        {  //Do Nothing:   }
//Produce a random item:
Buffer[in]  = nextProduction;
```

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 10 | | | | |

0:03:25 / 0:07:47


# Shared Memory Model

(Producer Consumer Example)

## Bounded Buffer – Shared Memory Solution


```
//shared data:
   int n=5, item, in ,out;
   int buffer[n];
//both the consumer and producer are
currently looking at buffer element 1.
   In = 2;      out = 1;
//Producer Process:
While(true)
{  while ( in+1 % n == out )
        {  //Do Nothing:   }
//Produce a random item:
Buffer[in]  = nextProduction;
```

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 10 | 20 | | | |


0:03:25 / 0:07:47


# Shared Memory Model

(Producer Consumer Example)

## Bounded Buffer – Shared Memory Solution

```
//shared data:
   int n=5, item, in ,out;
   int buffer[n];
//both the consumer and producer are
currently looking at buffer element 1.
   In = 5;     out = 1;
//Producer Process:
While(true)
{  while ( in+1 % n == out )
        {  //Do Nothing:   }
//Produce a random item:
Buffer[in]  = nextProduction;
```

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 |

0:03:25 / 0:07:47

21°C Cloudy 9:21 AM 10/7/2022

# Shared Memory Model

(Producer Consumer Example)

## Bounded Buffer – Shared Memory Solution

```
//shared data:
   int n=5, item, in ,out;
   int buffer[n];
//both the consumer and producer are
currently looking at buffer element 1.
   In = 5;     out = 1;
//Producer Process:
While(true)
{  while ( in+1 % n == out )
        {  //Do Nothing:   }
//Produce a random item:
Buffer[in]  = nextProduction;
```

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 |

```
//Consumer Process:
While(true)
{  while ( in == out )
        {   //Do Nothing:
        }
nextConsumed = buffer[out];
out = out +1 % n;
}
```

0:03:25 / 0:07:47

21°C Cloudy 9:22 AM 10/7/2022