#### Producer Consumer Problem

# Aim:

To implement a solution for Producer consumer problem using Semaphores.

# Description:

**Define a Shared Data Structure**: Establish a buffer of fixed size to facilitate communication between producers and consumers. This buffer serves as the intermediary storage for items produced by producers and consumed by consumers, allowing for efficient data exchange.

**Utilize Synchronization Primitives:** Employ semaphores, such as empty, full, and mutex, to manage access to the shared buffer and synchronize the actions of producers and consumers. The empty semaphore tracks the number of empty slots in the buffer, full indicates the number of filled slots, and mutex ensures mutual exclusion to prevent data corruption.

**Producer Logic:** Producers follow a protocol where they first wait on the empty semaphore, ensuring there is available space in the buffer to deposit new items. Once space is available, they enter a critical section protected by the mutex semaphore, where they deposit items into the buffer. After depositing an item, they signal the full semaphore to notify consumers of the availability of new data.

**Consumer Logic**: Consumers adhere to a similar protocol, where they first wait on the full semaphore, ensuring there are items available for consumption in the buffer. Once an item is available, they enter a critical section protected by the mutex semaphore, where they retrieve items from the buffer for processing. After consuming an item, they signal the empty semaphore to inform producers that space is available for new items.

**Initialization and Termination**: Begin by initializing the semaphores (empty, full, mutex) and creating producer and consumer threads/processes. During execution, these threads/processes interact with the shared buffer according to the defined protocols. Finally, ensure proper termination and cleanup procedures to release allocated resources and maintain system integrity.

**Algorithm:-**

Initialize semaphore 'empty' with BUFFER\_SIZE

Initialize semaphore 'full' with 0

Set in and out to 0

Declare buffer array of size BUFFER\_SIZE

Initialize mutex 'mutex'

**Algorithm Main():**

Start

Declare array pno[5],cno[5]

Declare array a[] that stores index of each producer and consumer

For i = 0 to 4 do

Create thread pno[i] running producer with argument a[i]

End For

For i = 0 to 4 do

Create thread cno[i] running consumer with argument a[i]

End For

For i = 0 to 4 do

Wait for thread pno[i] to finish

End For

For i = 0 to 4 do

Wait for thread cno[i] to finish

End For

Destroy mutex variable

Destroy semaphores empty and full 35

Stop

**Algorithm Producer ( pno ) :**

Start

For i = 0 to MAXITEMS-1 do

Wait(empty)

generate a random number to item

lock(mutex)

buffer[in] = item

Display "Producer pno: Produced Item item at in"

in= in +1

unlock(mutex)

signal(full)

End For

Return

**Algorithm Consumer ( cno ) :**

Start

For i = 0 to MAXITEMS-1 do

Wait(full)

Lock(mutex)

item = buffer[out]

Dispaly "Consumer cno: Consumed Item ‘item’ at out"

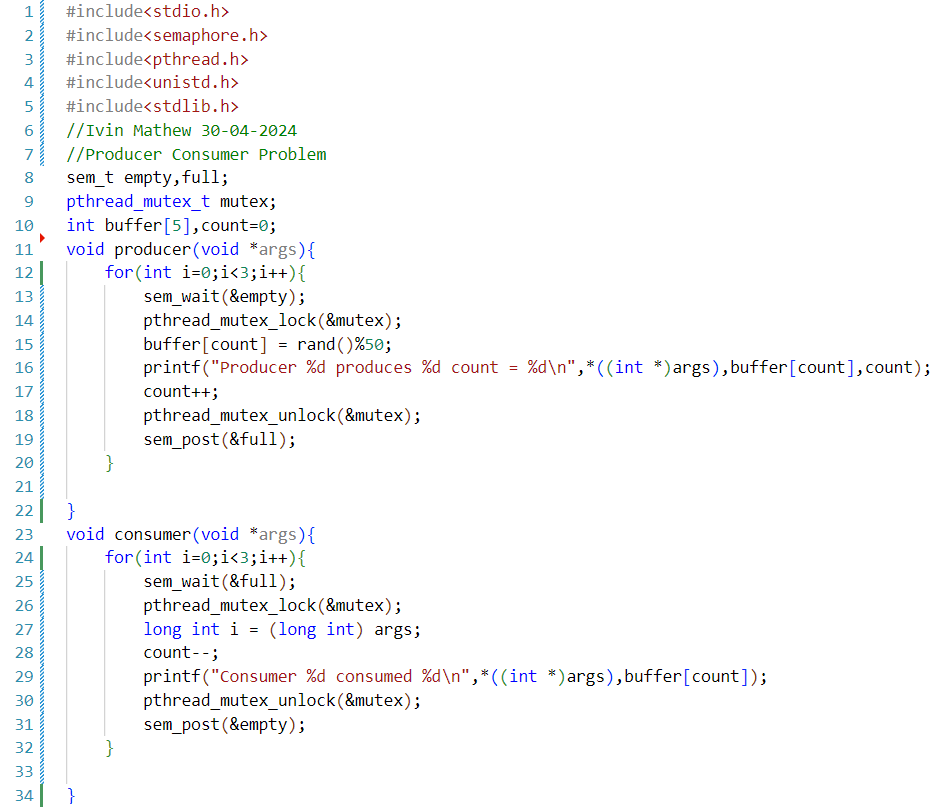
Unlock(mutex)

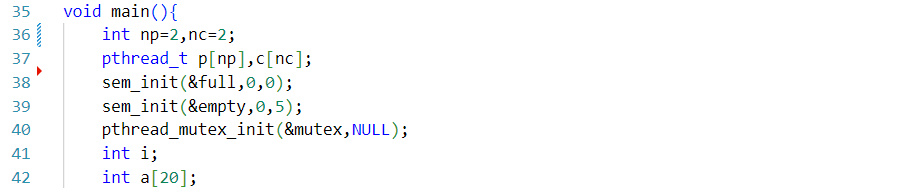
signal(empty)

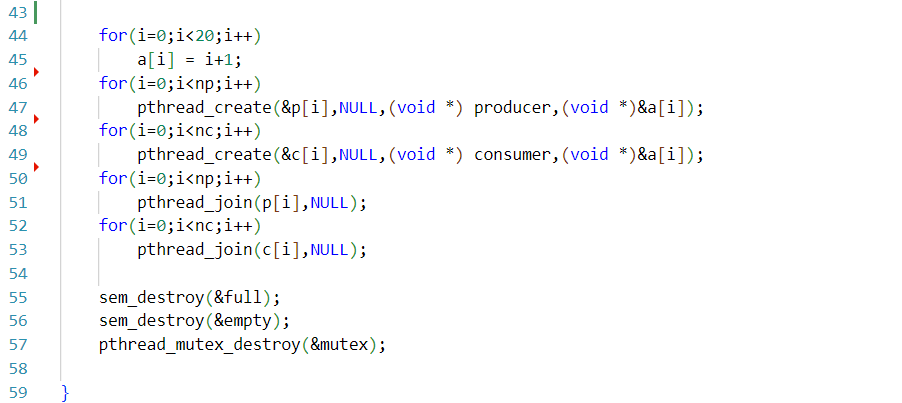
End For

Return

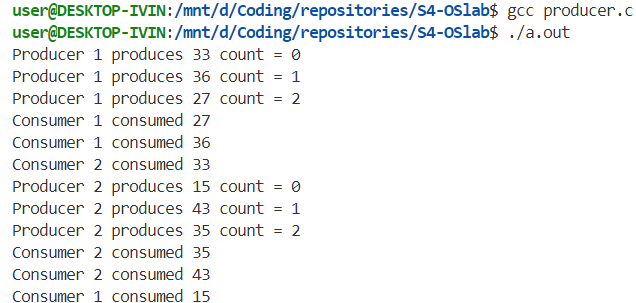
# Code







# Sample Output



# Result

The program has been executed and output has been verified