Notes- Producer-Consumer Problem (Bounded Buffer):

- Synchronization between the producer and consumer

- It ensures that the producer doesn't insert data when the buffer is full, and the consumer doesn't pick/remove data when the buffer is empty.

Semaphores Used:

- mutex (Binary Semaphore): Ensures exclusive access to the buffer.

- empty (Counting Semaphore): Tracks available empty slots in the buffer.

- full (Counting Semaphore): Tracks filled slots in the buffer.

Producer:

```
do {

wait(empty);   // Wait until there's an empty slot (empty > 0), then    decrement empty.

    wait(mutex);     // Acquire lock on buffer.

              // Critical Section: Add data to the buffer.

    signal(mutex);             // Release lock.

    signal(full);     // Increment full to indicate a filled slot.

} while (true);
```

Consumer:

```
do {
    wait(full);        // Wait until there's a filled slot (full > 0), then decrement full.

    wait(mutex);     // Acquire lock on buffer.

                                  // Critical Section: Remove data from the buffer.

    signal(mutex);   // Release lock.

    signal(empty);   // Increment empty to indicate an empty slot.

} while (true);
```

Important Points:

- The mutex semaphore ensures that only one producer or consumer can access the buffer at a time, preventing conflicts.

- empty semaphore represents the number of available empty slots, and full represents the number of filled slots.

- The wait operation on a counting semaphore decrements its value, and signal operation increments it.

- The use of semaphores helps in achieving synchronization and avoiding race conditions.

- This solution prevents the producer from adding data to a full buffer and the consumer from removing data from an empty buffer.

- The do-while(true) loop signifies that the producer and consumer processes continue their operations indefinitely.