Theory:

- The **Producer-Consumer Problem** is a classic synchronization problem.
- It involves a **fixed-size buffer** shared between a **Producer** (which produces items and places them in the buffer) and a **Consumer** (which removes and consumes items).
- The **buffer is the critical section**, requiring synchronization to prevent simultaneous access by both Producer and Consumer.

Synchronization Using Semaphores:

To solve this problem, we use two **counting semaphores**:

1. **Full**: Tracks the number of filled slots in the buffer.
2. **Empty**: Tracks the number of unoccupied slots in the buffer.
3. **Mutex**: Ensures mutual exclusion during buffer access.

Semaphore Operations:

- **wait() (P operation)**: Decreases the semaphore value by 1.
- **signal() (V operation)**: Increases the semaphore value by 1.

```
wait(S) {
    while(S <= 0);  // Busy waiting
    S--;
}

signal(S) {
    S++;
}
```

Semaphore Initialization:

```
mutex = 1;    // Ensures mutual exclusion
full = 0;     // Initially, no filled slots
empty = n;    // All slots are initially empty (n = buffer size)
```

Solution for Producer:

```
do {
    // Produce an item
    wait(empty);
    wait(mutex);
    // Place item in buffer
    signal(mutex);
    signal(full);
} while (true);
```

Explanation:

1. **Decrement** `empty` (since an item is being added to the buffer).
2. **Acquire** `mutex` to ensure exclusive access.
3. **Place item in buffer**.
4. **Release** `mutex`, allowing the consumer to access the buffer.
5. **Increment** `full` (since an item has been added).

Solution for Consumer:

```
do {
    wait(full);
    wait(mutex);
    // Remove item from buffer
    signal(mutex);
    signal(empty);
    // Consume item
} while (true);
```

Explanation:

1. **Decrement** `full` (since an item is being removed from the buffer).
2. **Acquire** `mutex` to ensure exclusive access.
3. **Remove item from buffer**.
4. **Release** `mutex`, allowing the producer to access the buffer.
5. **Increment** `empty` (since an item has been removed).

C Program for Producer-Consumer Problem:

```c
#include <stdio.h>
#include <stdlib.h>

int mutex = 1, full = 0, empty = 3, x = 0;

void producer();
void consumer();
int wait(int);
int signal(int);

int main() {
    int n;
    printf("\n1. Producer\n2. Consumer\n3. Exit");
    while (1) {
        printf("\nEnter your choice: ");
        scanf("%d", &n);
        switch (n) {
            case 1:
                if ((mutex == 1) && (empty != 0))
                    producer();
                else
                    printf("Buffer is full!!");
                break;
            case 2:
                if ((mutex == 1) && (full != 0))
                    consumer();
                else
                    printf("Buffer is empty!!");
                break;
            case 3:
                exit(0);
                break;
        }
    }
    return 0;
}

int wait(int s) {
    return (--s);
}

int signal(int s) {
    return (++s);
}

void producer() {
```

```
    mutex = wait(mutex);
    full = signal(full);
    empty = wait(empty);
    x++;
    printf("\nProducer produces the item %d", x);
    mutex = signal(mutex);
}

void consumer() {
    mutex = wait(mutex);
    full = wait(full);
    empty = signal(empty);
    printf("\nConsumer consumes item %d", x);
    x--;
    mutex = signal(mutex);
}
```