3. Develop a C program to simulate producer-consumer problem using semaphores

ALGORITHM:

1. Initialize the empty semaphore to the size of the buffer and the full semaphore to 0.
2. The producer acquires the empty semaphore to check if there are any empty slots in the buffer. If thereare no empty slots, the producer blocks until a slot becomes available.
3. The producer acquires the mutex to access the buffer, inserts a data item into an empty slot in thebuffer, and releases the mutex.
4. The producer releases the full semaphore to indicate that a slot in the buffer is now full.
5. The consumer acquires the full semaphore to check if there are any full slots in the buffer. If there areno full slots, the consumer blocks until a slot becomes available.
6. The consumer acquires the mutex to access the buffer, reads a data item from a full slot in the buffer,and releases the mutex.
7. The consumer releases the empty semaphore to indicate that a slot in the buffer is now empty.

CODE:

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

#define BUFFER_SIZE 3  // Size of the buffer

int mutex = 1;       // Mutex for critical section
int full = 0;        // Number of filled slots
int empty = BUFFER_SIZE; // Number of empty slots
int buffer[BUFFER_SIZE];  // Buffer array
int in = 0;          // Index for producer
int out = 0;         // Index for consumer
```

```
// Function prototypes
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;
        default:
            printf("Invalid choice!!");
            break;
      }
   }
   return 0;
}

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

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

void producer() {
   mutex = wait(mutex);          // Enter critical section
   full = signal(full);          // Increment full
   empty = wait(empty);          // Decrement empty

   // Produce an item
   buffer[in] = rand() % 100;    // Produce a random item
```

```
    printf("\nProducer produces the item %d at index %d", buffer[in], in);
    in = (in + 1) % BUFFER_SIZE;      // Circular increment

    mutex = signal(mutex);          // Exit critical section
}

void consumer() {
    mutex = wait(mutex);            // Enter critical section
    full = wait(full);              // Decrement full
    empty = signal(empty);          // Increment empty

    // Consume an item
    int item = buffer[out];
    printf("\nConsumer consumes item %d from index %d", item, out);
    out = (out + 1) % BUFFER_SIZE;    // Circular increment

    mutex = signal(mutex);          // Exit critical section
}
```

OUTPUT:

```
1. Producer
2. Consumer
3. Exit
Enter your choice: 1

Producer produces the item 83 at index 0
Enter your choice: 1

Producer produces the item 86 at index 1
Enter your choice: 1

Producer produces the item 77 at index 2
Enter your choice: 1
```

Buffer is full!!
Enter your choice: 2

Consumer consumes item 83 from index 0
Enter your choice: 2

Consumer consumes item 86 from index 1
Enter your choice: 2

Consumer consumes item 77 from index 2
Enter your choice: 2
Buffer is empty!!
Enter your choice: 3

=== Code Execution Successf**ul ===**