Shayan
DT-22037

OS LAB 04

Question 1: Implement the above code and paste the screen shot of the output.

Solution:

```
#include <stdio.h>

int main()
{
    int buffer[10], bufsize, in, out, produce, consume, choice = 0;
    in = 0;
    out = 0;
    bufsize = 10;

    while (choice != 3)
    {
        printf("\n1. Produce\t2. Consume\t3. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            if ((in + 1) % bufsize == out)
            {
                printf("\nBuffer is Full");
            }
            else
            {
                printf("\nEnter the value: ");
                scanf("%d", &produce);
                buffer[in] = produce;
                in = (in + 1) % bufsize;
            }
            break;

        case 2:
            if (in == out)
            {
                printf("\nBuffer is Empty");
            }
            else
            {
                consume = buffer[out];
                printf("\nThe consumed value is %d", consume);
```

Shayan
DT-22037

```
                out = (out + 1) % bufsize;
            }
            break;

        case 3:
            printf("\nExiting...\n");
            break;

        default:
            printf("\nInvalid choice! Please try again.");
        }
    }

    return 0;
}
```

```
1. Produce      2. Consume      3. Exit
Enter your choice: 1

Enter the value: 2

1. Produce      2. Consume      3. Exit
Enter your choice: 2

The consumed value is 1
1. Produce      2. Consume      3. Exit
Enter your choice: 2

The consumed value is 2
1. Produce      2. Consume      3. Exit
Enter your choice: 2

Buffer is Empty
1. Produce      2. Consume      3. Exit
Enter your choice: 3

Exiting...
```

Question 2: Solve the producer-consumer problem using linked list. (You can perform this task using any programming language)

Note: Keep the buffer size to 10 places.

Shayan
DT-22037

Solution:

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

typedef struct Node
{
    int data;
    struct Node *next;
} Node;

Node *head = NULL;
pthread_mutex_t mutex;

void produce()
{
    int value;
    printf("\nEnter value to produce: ");
    scanf("%d", &value);

    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode)
    {
        printf("\nMemory allocation failed!");
        return;
    }
    newNode->data = value;
    newNode->next = NULL;

    pthread_mutex_lock(&mutex);

    if (head == NULL)
    {
        head = newNode;
    }
    else
    {
        Node *temp = head;
        while (temp->next)
        {
            temp = temp->next;
        }
        temp->next = newNode;
    }

    printf("Produced: %d\n", value);
```

```
    pthread_mutex_unlock(&mutex);
}

void consume()
{
    pthread_mutex_lock(&mutex);

    if (head == NULL)
    {
        printf("\nBuffer is empty, nothing to consume.");
    }
    else
    {
        Node *temp = head;
        printf("\nConsumed: %d", temp->data);
        head = head->next;
        free(temp);
    }

    pthread_mutex_unlock(&mutex);
}

void displayBuffer()
{
    pthread_mutex_lock(&mutex);

    if (head == NULL)
    {
        printf("\nBuffer is empty.");
    }
    else
    {
        printf("\nCurrent Buffer: ");
        Node *temp = head;
        while (temp)
        {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL");
    }

    pthread_mutex_unlock(&mutex);
}

int main()
{
```

```
    int choice;
    pthread_mutex_init(&mutex, NULL);

    while (1)
    {
        printf("\n1. Produce");
        printf("\n2. Consume");
        printf("\n3. Display Buffer");
        printf("\n4. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            produce();
            break;
        case 2:
            consume();
            break;
        case 3:
            displayBuffer();
            break;
        case 4:
            printf("\nExiting...\n");
            pthread_mutex_destroy(&mutex);
            return 0;
        default:
            printf("\nInvalid choice! Please try again.");
        }
    }
}
```

Shayan
DT-22037

```
1. Produce
2. Consume
3. Display Buffer
4. Exit
Enter your choice: 1

Enter value to produce: 10
Produced: 10

1. Produce
2. Consume
3. Display Buffer
4. Exit
Enter your choice: 1

Enter value to produce: 20
Produced: 20

1. Produce
2. Consume
3. Display Buffer
4. Exit
Enter your choice: 3

Current Buffer: 10 -> 20 -> NULL
```

```
1. Produce
2. Consume
3. Display Buffer
4. Exit
Enter your choice: 2

Consumed: 10
1. Produce
2. Consume
3. Display Buffer
4. Exit
Enter your choice: 2

Consumed: 20
1. Produce
2. Consume
3. Display Buffer
4. Exit
Enter your choice: 4

Exiting...
```

Shayan
DT-22037

Question 3: In producer-consumer problem what difference will it make if we utilize stack for the buffer rather than an array?

Solution:

In the producer-consumer problem, using a **stack** instead of an **array (queue)** would fundamentally change the way items are produced and consumed. A **stack** follows a **Last-In, First-Out (LIFO)** order, meaning the most recently produced item is consumed first. This is different from an **array-based queue**, which follows a **First-In, First-Out (FIFO)** approach, where the oldest item is consumed first. The **LIFO** behavior of a stack could be beneficial in scenarios where newer data is more relevant (e.g., caching or backtracking algorithms). However, in typical producer-consumer scenarios like job scheduling or message processing, **FIFO queues are preferred** because they ensure fairness and prevent starvation of older items. Additionally, with a stack, the consumer may never get to the older items if new ones keep being produced rapidly, potentially leading to data loss in time-sensitive applications.