Noman Ahmed
DT-22032

Operating System (CT-353)

Lab no 04

Noman Ahmed
DT-22032

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

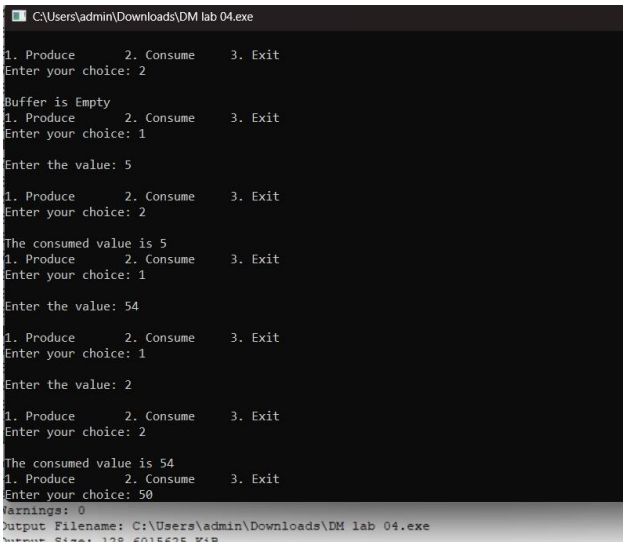
```
#include <stdio.h>

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

   while (choice != 3) {
      printf("\n1. Produce \t 2.
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;
```

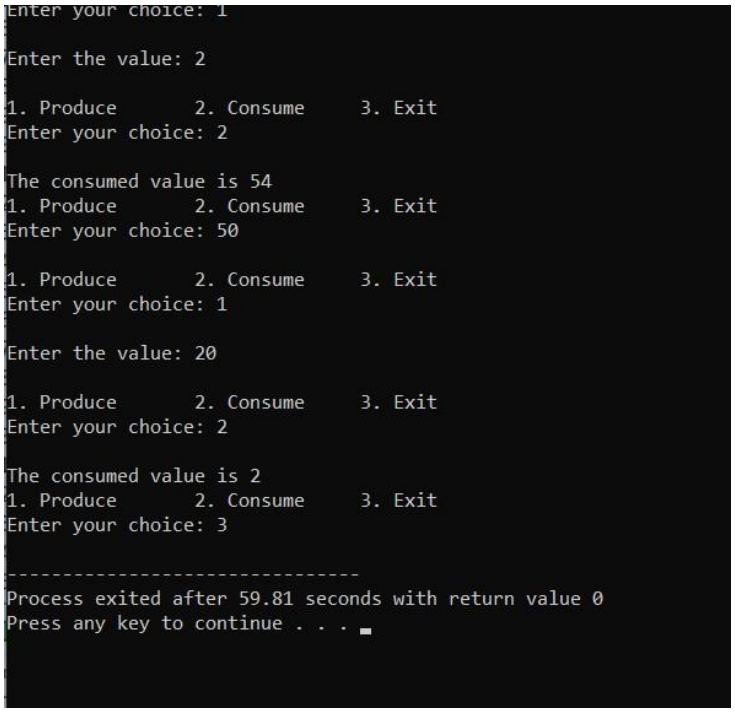
Noman Ahmed
DT-22032

```
            }
break;

case 2:
            if (in == out)
               printf("\nBuffer is
Empty");              else {
               consume = buffer[out];
               printf("\nThe consumed value is %d", consume);
               out = (out + 1) % bufsize;
            }
break;
      }
   }
}
```

Noman Ahmed
DT-22032

OUTPUT :





Noman Ahmed
DT-22032

2) Solve the producer-consumer problem using linked list. Note: Keep the buffer size to 10 places.

```
#include <stdio.h>

#define BUFFER_SIZE 10

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

Node* head =
NULL; Node*
tail = NULL; int
count = 0;

pthread_mutex_t mutex;
sem_t empty, full;

void insert(int item) {
   Node* newNode =
(Node*)malloc(sizeof(Node));
newNode->data = item;
   newNode->next = NULL;
```

Noman Ahmed
DT-22032

```
   if (tail == NULL) {
head = tail =
newNode;
   } else {
      tail->next = newNode;
      tail = newNode;
   }
   count++;
}

int remove_item() {
   if (head == NULL) return -1;

   Node* temp =
head;    int item
= temp->data;
   head = head->next;

   if (head == NULL) tail = NULL;


free(temp);
count--;
return
item;
}

void*
producer(void*
arg) {    int item;
```

Noman Ahmed
DT-22032

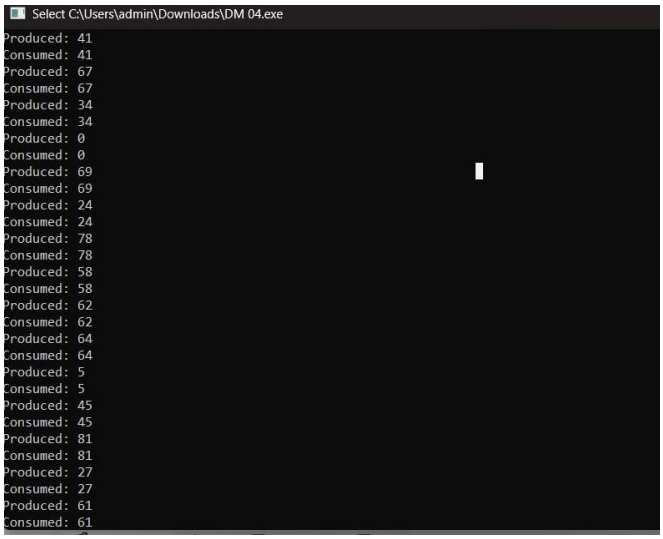
```
while (1) {
item = rand() %
100;
sem_wait(&empty
);
      pthread_mutex_lock(&mutex);

      insert(item);
      printf("Produced: %d\n", item);

      pthread_mutex_unlock(&mutex);
sem_post(&full);
      sleep(1);
   }
}

void* consumer(void* arg) {
   int item;
while (1) {
sem_wait(&fu
ll);
      pthread_mutex_lock(&mutex);

      item = remove_item();
      printf("Consumed: %d\n", item);

      pthread_mutex_unlock(&mutex);
sem_post(&empty);
      sleep(1);
   }
```

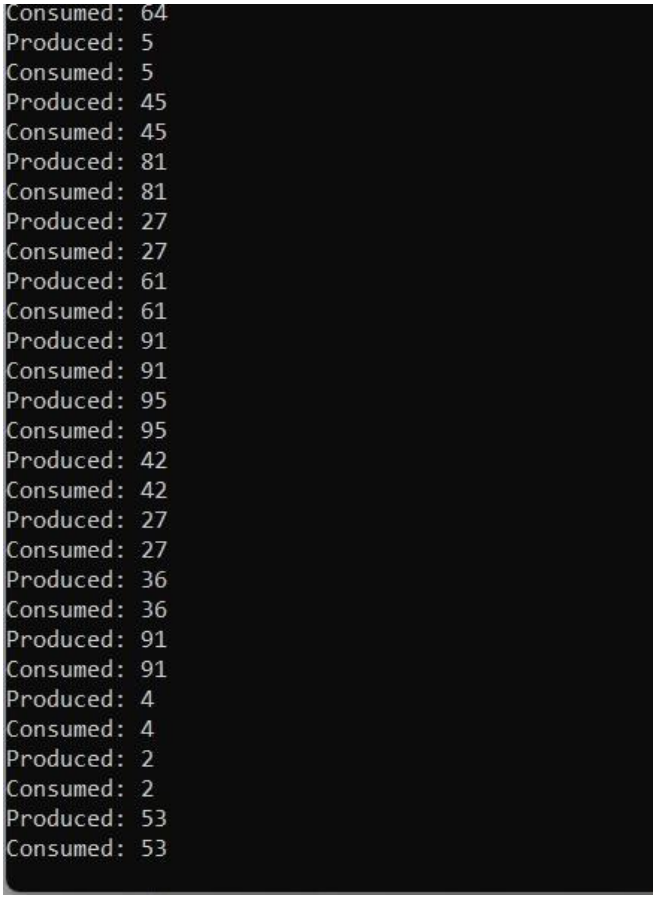
Noman Ahmed
DT-22032

```
}

int main() {
   pthread_t prod, cons;

   pthread_mutex_init(&mutex,
NULL);     sem_init(&empty, 0,
BUFFER_SIZE);
   sem_init(&full, 0, 0);

   pthread_create(&prod, NULL,
producer, NULL);
   pthread_create(&cons, NULL,
consumer, NULL);

   pthread_join(prod, NULL);
   pthread_join(cons, NULL);


pthread_mutex_destroy(&mut
ex);     sem_destroy(&empty);
sem_destroy(&full);
   return 0;
}
```

Noman Ahmed
DT-22032

Output:





Noman Ahmed
DT-22032

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

Using a stack instead of a queue in the producer-consumer problem fundamentally changes the processing order from FIFO to LIFO, which may not be suitable for many traditional producer-consumer use cases.