**Operating System**

**(CT-353)**

Lab no 04

**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;

} 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;

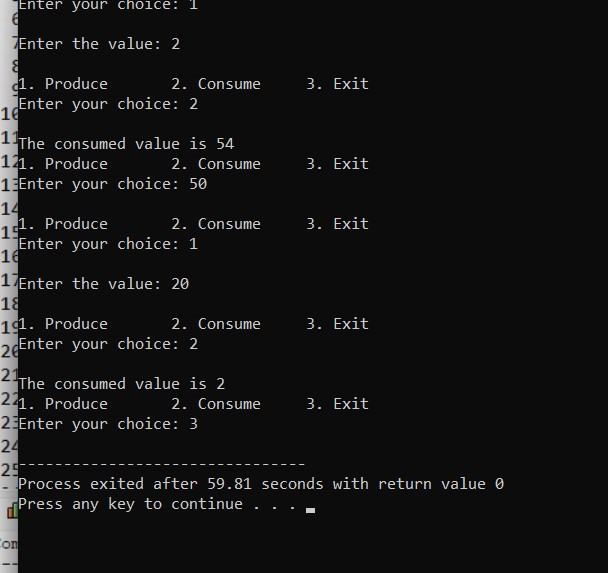
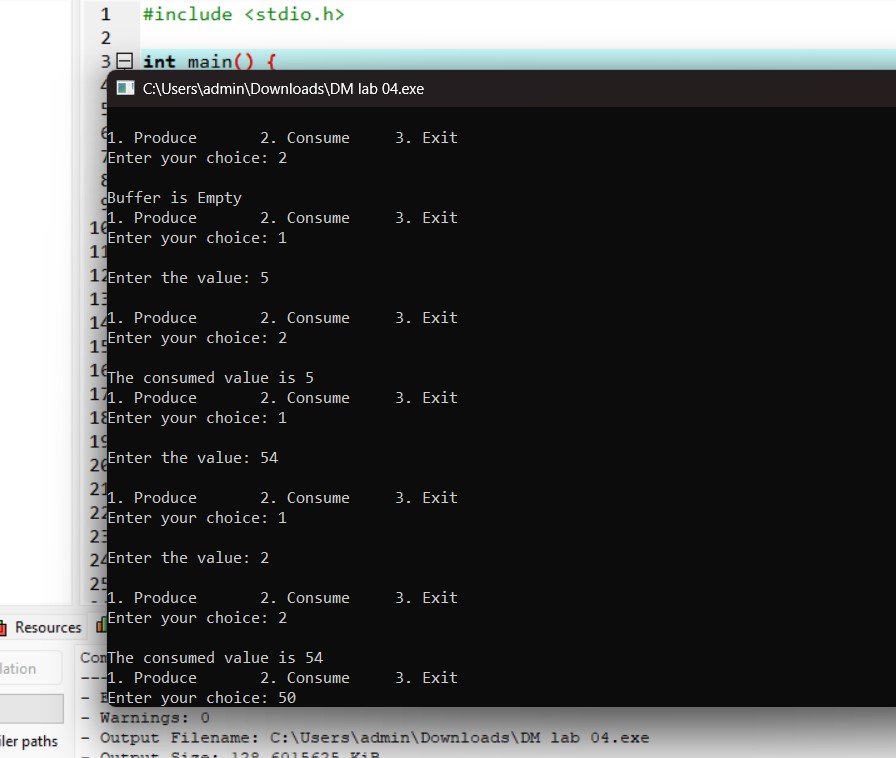
} break;

}

}

}

Output :



**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;

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; 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(&full);

pthread\_mutex\_lock(&mutex);

item = remove\_item();

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

pthread\_mutex\_unlock(&mutex); sem\_post(&empty);

sleep(1);

}

}

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);

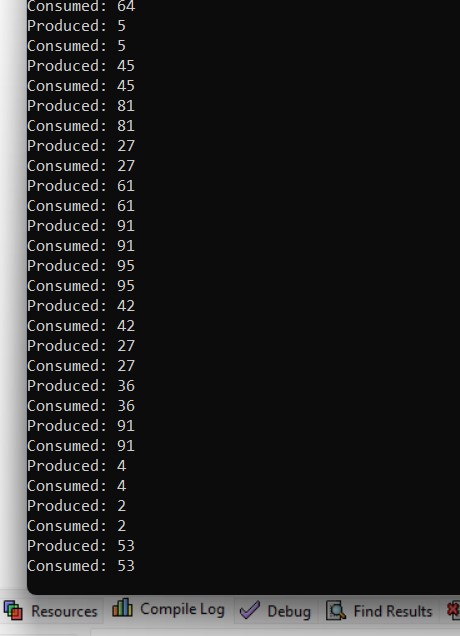
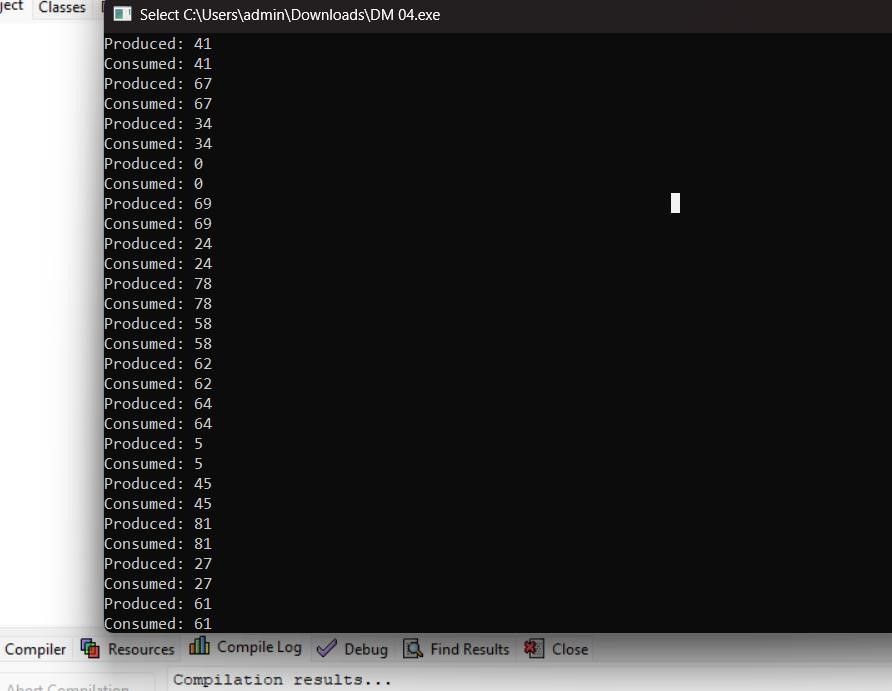
pthread\_join(cons, NULL);

pthread\_mutex\_destroy(&mutex); sem\_destroy(&empty); sem\_destroy(&full);

return 0;

}

**Output:**



**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.