**Name:**B. Nageswar

**Reg-No**: 192372005

18.Construct a C program to simulate producer-consumer problem using semaphores.

**Aim:**

The aim of the program is to simulate the producer-consumer problem using semaphores. The producer creates data and puts it in a buffer, while the consumer consumes the data from the buffer. Semaphores are used to synchronize access to the shared buffer.

**Algorithm:**

1. **Initialization:**
   * Initialize two semaphores: empty (to track the number of empty slots) and full (to track the number of full slots).
   * Initialize a mutex semaphore for mutual exclusion.
2. **Producer:**
   * Wait on the empty semaphore (to ensure there is space).
   * Wait on the mutex semaphore (for mutual exclusion).
   * Add an item to the buffer.
   * Signal the full semaphore (indicating a full slot).
   * Signal the mutex semaphore to release mutual exclusion.
3. **Consumer:**
   * Wait on the full semaphore (to ensure there is data).
   * Wait on the mutex semaphore (for mutual exclusion).
   * Consume an item from the buffer.
   * Signal the empty semaphore (indicating an empty slot).
   * Signal the mutex semaphore to release mutual exclusion.

**Procedure:**

* The producer creates data and puts it into the buffer.
* The consumer retrieves data from the buffer and consumes it.
* The semaphores ensure that the buffer is accessed in a synchronized manner.

### Code:

### #include <stdio.h>

### #include <stdbool.h>

### int main() {

### int n, m, i, j, k;

### printf("Enter number of processes: ");

### scanf("%d", &n);

### printf("Enter number of resources: ");

### scanf("%d", &m);

### int Allocation[n][m], Maximum[n][m], Need[n][m], Available[m];

### printf("Enter Allocation matrix:\n");

### for (i = 0; i < n; i++)

### for (j = 0; j < m; j++)

### scanf("%d", &Allocation[i][j]);

### printf("Enter Maximum matrix:\n");

### for (i = 0; i < n; i++)

### for (j = 0; j < m; j++)

### scanf("%d", &Maximum[i][j]);

### printf("Enter Available resources:\n");

### for (j = 0; j < m; j++)

### scanf("%d", &Available[j]);

### for (i = 0; i < n; i++)

### for (j = 0; j < m; j++)

### Need[i][j] = Maximum[i][j] - Allocation[i][j];

### bool Finish[n];

### for (i = 0; i < n; i++)

### Finish[i] = false;

### int SafeSequence[n], work[m];

### for (j = 0; j < m; j++)

### work[j] = Available[j];

### int count = 0;

### while (count < n) {

### bool found = false;

### for (i = 0; i < n; i++) {

### if (!Finish[i]) {

### for (j = 0; j < m; j++)

### if (Need[i][j] > work[j])

### break;

### if (j == m) {

### for (k = 0; k < m; k++)

### work[k] += Allocation[i][k];

### SafeSequence[count++] = i;

### Finish[i] = true;

### found = true;

### }

### }

### }

### if (!found) {

### printf("System is in an unsafe state.\n");

### return 0;

### }

### }

### printf("System is in a safe state.\nSafe sequence is: ");

### for (i = 0; i < n; i++)

### printf("%d ", SafeSequence[i]);

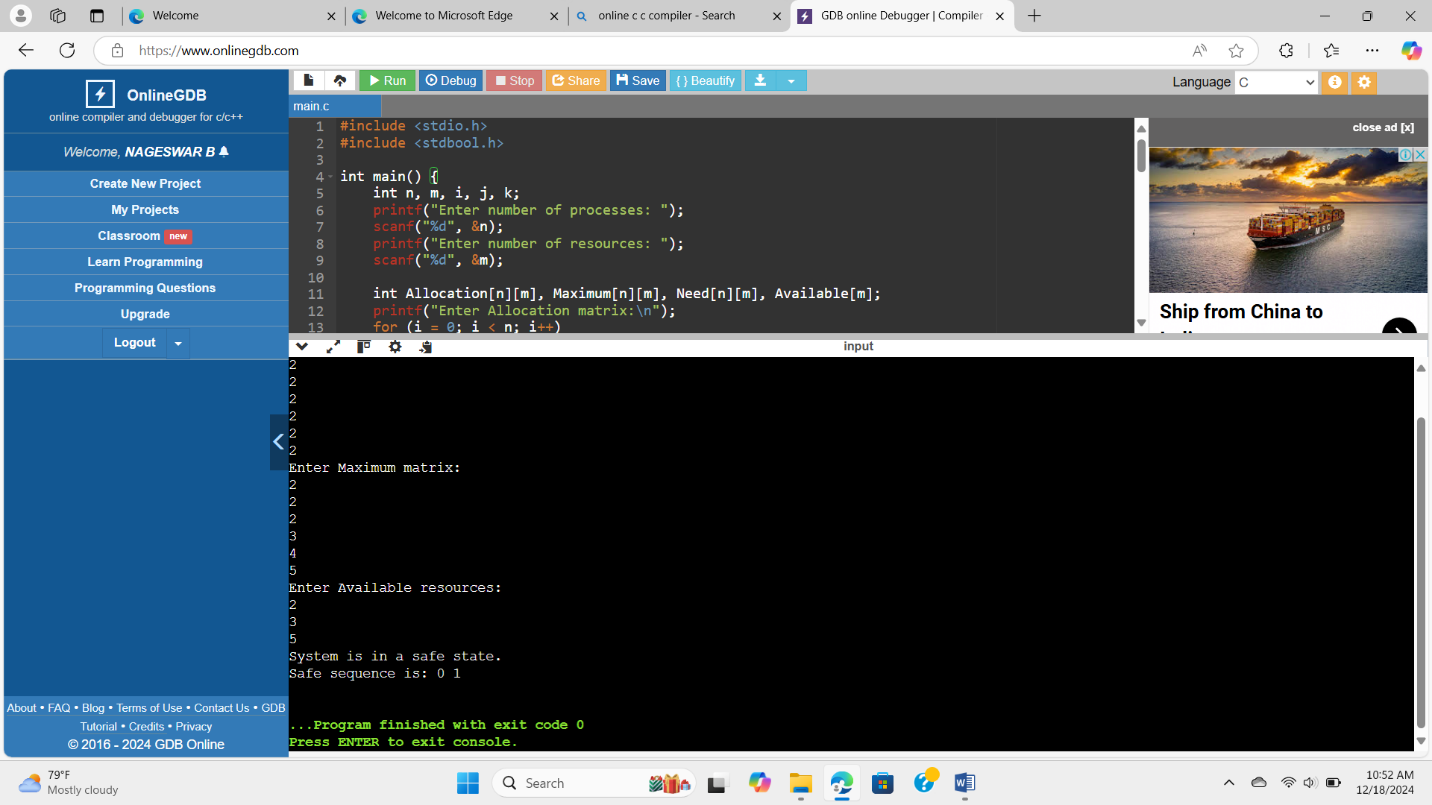
### printf("\n");

### return 0;

### }

**Result:**

* The producer generates random numbers and places them in the buffer.
* The consumer retrieves and consumes these numbers.
* The semaphores ensure that the producer and consumer operate without conflicts, and the buffer is accessed safely.

**Output:**